TU Clausthal

Clausthal University of Technology

Master Thesis

**Abram Lawendy**

# Domain Engineering as a Solution for Bridging the Gap Between Ontology Engineers and Domain Experts

Master Thesis at Institute for Software and Systems Engineering

First Censor: Prof. Dr. Andreas Rausch
Second Censor: Prof. Dr. Rüdiger Ehlers
Supervisor: M. Sc. Sebastian Lawrenz

Date of Delivery: December 4, 2019

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Master Thesis

**Domain Engineering as a Solution for Bridging the Gap Between Ontology Engineers and Domain Experts**

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass alle Stellen dieser Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht wurden und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegt wurde.

Des Weiteren erkläre ich, dass ich mit der öffentlichen Bereitstellung meiner Master Thesis in der Instituts- und/oder Universitätsbibliothek einverstanden bin.

This following english translation is for information purposes only.
The original german text above is the legally binding version.

*I hereby ensure that I am the sole drafter of the work and did not use any sources and aids other than those listed, and that all portions of this work literally or analogously taken from other sources were labelled as such, and that the work has not been submitted in the same or similar form to any other evaluating entity.*

*Furthermore, I declare that I am willing to have my thesis made publicly available in the institute and/or university library*

Clausthal-Zellerfeld, den December 4, 2019 (Abram Lawendy)

# Acknowledgment

I would first like to thank my thesis advisor Prof. Dr. Andreas Rausch, for his suggestions for improvement and his consultation and assessment of my thesis. I would also like to acknowledge Prof. Dr. Rüdiger Ehlers as the second assessor of this thesis for his valuable comments on this work.

I would like to thank all those who supported and motivated me during the preparation of my master thesis. Special thanks go to my supervisor Sebastian Lawrenz for the extensive personal support during the entire processing time. I owe particular thanks to Mirco Schindler, Helge Fischer, and Priyanka Sharma, who were always available for questions.

Last but not least, I would like to give my gratitude to my beloved family and my lovely fiancée. Special thanks go to my father, who is always watching over me.

# Summary

One of the main challenges in software systems development is reusability. The interaction between traditional software systems is taking place through their interfaces. Thus, as more software systems are developed, the complexity of the interconnection between the interfaces grows dramatically. This complexity resulted in a decrease in development time and code quality. The role of a subject-matter expert (SME) or *Domain Expert*, a person with knowledge in a specific area, emerged to tackle this problem and increase the software reusability by modeling the domain rather than the technology. It resulted in a collaboration between the domain experts and developers and sped up the software development life cycle. Similarly, the role of *DevOps* emerged to bridge the gap between developers and operators in order to automate the process between software development and production release.

The main challenge today is to achieve reusability and interconnections between domains of the same or different interests. The ontology concept was introduced as an engineering artifact to describe a reality semantically. Ontologies aim to represent information in a way to be understood and processed by a computer. Therefore, an *Ontology Engineer* is a person with knowledge for ontology's vocabulary, rules of inference, logic, and its construction. Common domain description languages often lack the semantic representation of their entities. By contrast, the semantic meaning of a domain of interest can be expressed using the ontology's vocabularies and axioms.

The inadequate conceptual knowledge representation of a domain expert and the inadequate domain knowledge of an ontology engineer yields a gap between both worlds. Thus, the usage of ontologies in the domain modeling is yet considered to be a challenge. *Domain Engineering* is an approach that aim for the creation and development of domains on semantic bases.

This work focus on investigating the challenges and limitations in the syntactic- and semantic-based development fields. The main objective is to approach the domain engineering as a solution for bridging the gap between domain experts and ontology engineers. Also, the study introduces a domain knowledge development life cycle approach to help by the creation and development of domain representations on semantic bases. The research is conducted in three segments. First by defining the problem and limitation, then conducting extensive literature and related work review to scrutinize the domain experts and ontology engineers roles based on some criteria in their scope of an intersection. After that, investigate existing approaches and tools to construct the domain knowledge life cycle toolchain. Finally, the study concludes the essential presence of both the domain knowledge, presented by the domain expert, and the formalization of semantic conceptualization, presented by the ontology engineer, in the domain engineering approach.

# Contents

# Abbreviations

**ADL** Architecture Description Language

**AHC** Agglomerative Hierarchical Clustering

**AM** AgreementMaker

**AML** AgreementMakerLight

**ASCII** American Standard Code for Information Interchange

**COMA** Combination of Matching algorithms

**CSV** comma-separated values

**DAG** directed acyclic graph

**DAiSI** Dynamic Adaptive System Infrastructure

**DDD** Domain-driven Design

**GUI** graphical user interface

**IDF** Inverse Document Frequency

**IR** information retrieval

**IRI** Internationalized Resource Identifier

**OCR** optical character recognition

**OWL** Web Ontology Language

**RDF** Resource Description Framework

**RDFS** Resource Description Framework Schema

**SME** subject-matter expert

**SPARQL** SPARQL Protocol and RDF Query Language

**SPL** software product line

**SQL** Structured Query Language

**SWOOP** SemanticWeb Ontology Overview and Perusal

**TF** Term Frequency

**URI** Uniform Resource Identifier

**XMI** XML Metadata Interchange

**XML** Extensible Markup Language

**XSD** XML Schema Definition

# List of Figures

# List of Tables

# 1 Introduction

> A domain is an area of knowledge that uses common concepts for describing phenomena, requirements, problems, capabilities, and solutions. [RBSC+13]

The domain can be specific to a company or a software product line (SPL). SPL is a set of software-based systems that satisfy some specific business needs. Middle to large size companies has a vast number of software systems, each with its own set of data stores. Even though the interaction between software systems is taking place with the help of their well-defined interfaces, as more software systems are developed, the complexity of the interconnection between the interfaces grows dramatically. [Eva14, RBSC+13, Ree13]

## 1.1 Motivation

Traditional software systems operated in the form of passing the data between one another periodically. This process is called batch mode and described as "tightly coupled" because any changes must be applied to all systems. This results in decreasing the development time and code quality, which is known by the bottleneck effect in software development. The role of *Domain Expert* emerged to approach this problem and introduce the Domain-driven Design (DDD) to create better software by modeling the domain rather than the technology. And because development is iterative, it resulted in collaboration between domain experts and developers, in other words, those who know the domain and those who know how to build the software. [Eva14, RBSC+13, Ree13]

Today's applications' main characteristic is openness, namely the interaction between applications does not occur only within their domain, but also between different domains. In his book "The Knowing Organization," Chun Wei Choo described an organization to be knowing in case it presents some information-based view. The organization uses this information and knowledge to adapt to external changes and internal growth. In his diagram 1.1, the progression of signals into knowledge is depicted. The "cognitive structuring," also known as "semantic representation," is the process of obtaining information from the data by assigning meaning regarding the domain. The role of *Ontology Engineer* emerged to address this field and introduce ontologies as semantic descriptions for domains. An ontology is an engineering artifact used to describe a certain reality semantically. It consists of a set of knowledge terms and axioms. Thus an *Ontology Engineer* is a person with knowledge for building ontologies. [Cho06, MS02, Hen01]

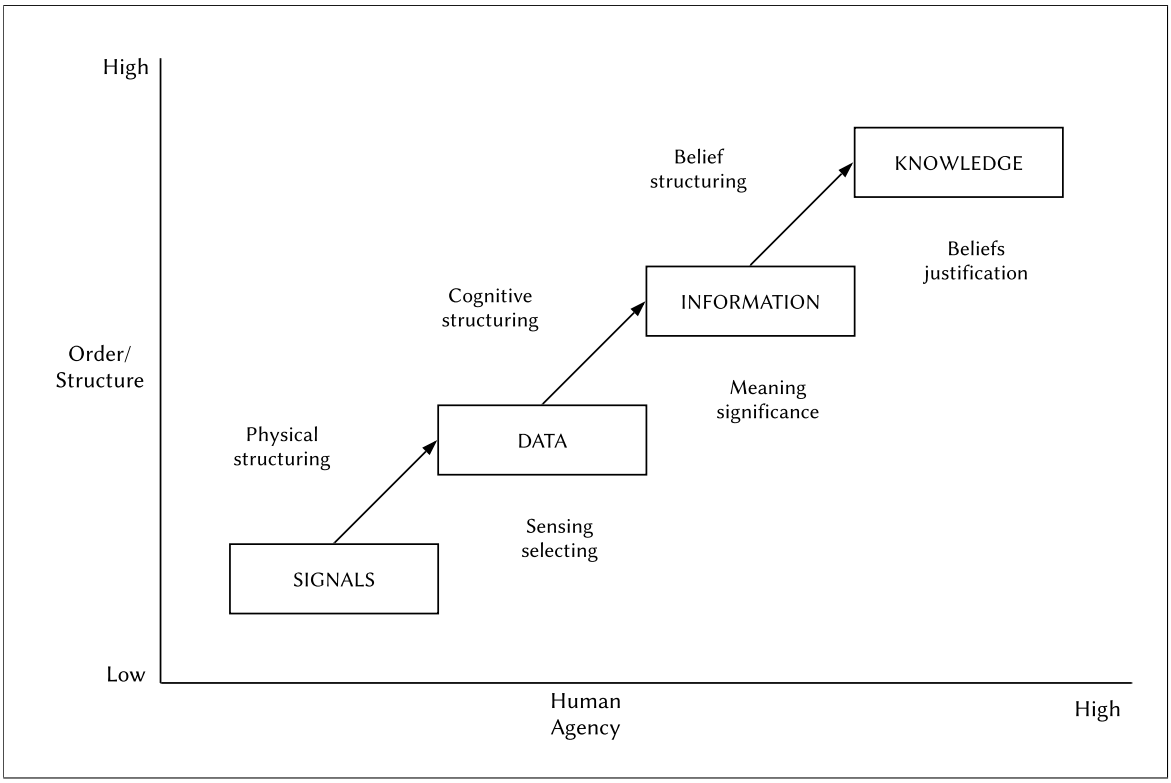


Figure 1.1: Data, information and knowledge, according to Choo [Cho06, p. 132]

## 1.2 Problem and Limitation

Reusability and increasing productivity are the main reasons behind introducing the different roles in the information systems and software engineering world. Whether it is domain experts and developers, developers and operators, or domain experts and ontology engineers, the main challenge is bridging the gap through aggregate information. Similarly, the *DevOps*, as a set of practices, was introduced to bridge the gap between the developers and the systems operators, in order to automate the process between software development and release or operation.

While ontology is an old concept, it is still not widespread in the software systems development field, due to its extensive composition process. One famous example of knowledge bases, which is another word for ontologies, is Wikidata. It is one of the largest free knowledge bases with 68,599,689 items[1]. When constructing a knowledge base for a specific domain of interest, domain experts possess limited knowledge for the formal language and logic that describes ontological concepts. On the other hand, ontology engineers do not fully grasp the domain as interpreted by a domain expert, and therefore they are unable to fully describe and reason over it. The role of *Domain Engineer* emerged to approach this problem and bridge the gap between the domain experts and ontology engineers, in other words, those who know the domain and those who know how to to describe it semantically.

[1]Number based on November 2019

## 1.3 Goal and Research Question

The main goal of this thesis is to approach the domain engineering as a solution for bridging the gap between the domain experts and ontology engineers. In addition, the study introduces a domain knowledge development life cycle approach to help the creation and development of domain representations on semantic bases. The work compares some tools and presents a toolchain for the proposed domain knowledge development life cycle. The creation and development of domain ontologies open a group of research questions. In the following, the ones this thesis focuses on.

- *Which core functionality is required for the development of domain knowledge on semantic bases?*
- *Is the presence of domain knowledge as well as semantic conceptualization knowledge, essential in the process of domain knowledge development?*
- *Is there a tool that makes one of both roles: domain expert and ontology engineer, dispensable?*

## 1.4 Structure of the Thesis

Besides this introduction chapter, this thesis is divided into six chapters. Chapter 2 reviews the literature and related work conducted in two well-known fields of study: the syntactic- and semantic-based development approaches. While the syntactic-based approach is used in the domain modeling development field, the semantic-based approach is used for the knowledge representation and ontology evolution field. Chapter 3 presents two related work conducted in the field of syntactic- and semantic-matching, respectively. Chapter 4 proposes a domain knowledge development procedure and compares existing approaches and tools which contribute to the development of domain knowledge. Chapter 5 details the result of the compared tools and argues the practicability of the proposed toolchain. Chapter 6 concludes the research conducted during this study, in addition to answering the opened research questions and presenting suggestions for future research works.

# 2 Background

Over the past years, reusability has always become a challenge. It evolved from considering a small chunk of source lines of code, to a complete process. The greatest challenge today lies in the reusability of an entire domain of interest. This chapter introduced two well-known research approaches: syntactic- and semantic-based integration. As mentioned before, the main research interest aims to investigate domain development on semantic bases. A critical component in the semantic representation of a domain is the adaptation to changes. Therefore, this chapter reviews the integration concept to help understand the challenges and limitations in each approach.

## 2.1 Syntactic-based Integration

The syntactic-based integration is the process of finding similarity by the interpretation of the input regarding its structure. Using syntactic matching such as string-based matching algorithms, a similarity measure can be calculated to decide whether two entities share a similar structure and therefore meaning. In some cases, a syntactic-based similarity measures can be impractical and unreliable. For example, Blu-ray disc, BD, B.D., and BD-Rom can be considered equivalent in some contexts. However, in some other context, BD may mean Birthday and in some other Business development. Levenshtein (1965) introduced *Edit Distance*, a widely-used similarity measure to compare two string. The algorithm determines whether two strings are alike by calculating how many atomic actions are required to transform one string into the other one. The atomic actions are addition, deletion, and replacement of string's characters. Luckily, in most cases, syntactic variations of the same entity often share nearby spellings and abbreviations.
[ES13, Ehr07]

### String Matching

The string matching procedure aims to find strings that refer to the same entities. The procedure is used in data integration tasks, including schema and data matching, and information extraction. The procedure computes a similarity score between two given strings. The matched strings can be merged or linked to improve data quality and provide more comprehensive information. For example, the strings Alan Mathison Turing and Alan M. Turing may refer to the same person. The procedure has two challenges: accuracy and scalability. The computed similarity score would deliver an unsatisfying result in cases like typing and optical character recognition (OCR) errors, or custom abbreviation. Moreover, data sources are unlikely to contain semantic information to determine whether two strings

refer to the same entity.
[DHI12, Dat10]

### Similarity Measures Matching Techniques

Similarity measures operate by mapping a pair of strings $(x, y)$ into a number in the range $[0, 1]$. The higher the score, the more likely that $x$ and $y$ matches. This can be represented by the following equation: $s(x, y) \geq t$, where $t$ is a predefined threshold. A threshold is a value at or above which the pair of strings are considered to be similar. Table 2.1 presents an example of matching person names using similarity measures techniques. Applying similarity measure to all pairs of strings in two data sets $X$ and $Y$ is very costly $O(|X||Y|)$ and therefore, impractical. This scalability problem can be overcome using a solution called *blocking solution*. The procedure calls a method FindCands that finds all strings in a data set $Y$ that may match a given string $x$. It takes $O(|X||Z|)$, where $Z$ is the so-called *umbrella set* of $x$, obtained from the FindCands method. FindCands uses techniques based on indexing or filtering heuristics to find all potential strings to match $x$.
[DHI12]

| Set $X$ | Set $Y$ | Matches |
|---|---|---|
| $x_1$ = Dave Smith | $y_1$ = David D. Smith | $(x_1, y_1)$ |
| $x_2$ = Joe Wilson | $y_2$ = Daniel W. Smith | $(x_3, y_2)$ |
| $x_3$ = Dan Smith | | |
| (a) | (b) | (c) |

Table 2.1: An example of matching person names, according to Doan et al. [DHI12, p. 96]

### Sequence-based Similarity Measures Techniques

The sequence-based similarity measures treat the strings as sequences of characters. It computes the cost of transforming one string to the other. The smaller the cost, the more likely the two strings match. *The Needleman-Wunch Measure* and *The Smith-Waterman Measure* are two very famous sequence-based similarity measures algorithms.
[DHI12]

#### *The Needleman-Wunch Measure*

The Needleman-Wunch measure belongs to the sequence-based similarity measures. It generalizes the Levenshtein distance described in section 2.1. The arithmetic formula is shown in equation 2.1. The measure calculates an alignment between two strings based on a set of correspondences between their characters. The score of the alignment is the sum of the scores of all correspondences in the alignment, minus the penalties for gaps.

[DHI12]

$$
\begin{aligned}
s(i,j) &= max \begin{cases} s(i-1,j-1) + c(x_i, y_j) \\ s(i-1,j) - c_g \\ s(i,j-1) - c_g \end{cases} \\
s(0,j) &= -jc_g \\
s(i,0) &= -ic_g
\end{aligned} \tag{2.1}
$$

*The Smith-Waterman Measure*

The Smith-Waterman measure also belongs to the sequence-based similarity measures. Different to the Needleman-Wunch, this measure finds two substrings of $x$ and $y$ that are most likely to be similar, then return the score computed as the score for $x$ and $y$. The matching computation can start at any position in the strings, which helps to ignore prefixes or suffixes. The arithmetic formula is shown in equation 2.2. Consider the two strings (Prof. Max S. Mustermann, TU Clausthal) and (Max S. Mustermann, Professor). The measure computes a local alignment matching by ignoring particular prefixes, e.g., Prof., and suffixes, e.g., TU Clausthal.
[DHI12]

$$
\begin{aligned}
s(i,j) &= max \begin{cases} 0 \\ s(i-1,j-1) + c(x_i, y_j) \\ s(i-1,j) - c_g \\ s(i,j-1) - c_g \end{cases} \\
s(0,j) &= 0 \\
s(i,0) &= 0
\end{aligned} \tag{2.2}
$$

### Set-based Similarity Measures Techniques

The set-based similarity measures treat the string as sets of generated tokens. To generate tokens from strings, the space character and common stop words, (e.g., the, and, of), are excluded and used as delimiters. Another common type of tokens is *q-grams*, which divides the string into substrings of length $q$. For example, the set of all 3-grams of max mustermann is {##m, #ma, max, …, ann, nn#, n##}.
[DHI12]

*The TF/IDF Measure*

The Term Frequency/Inverse Document Frequency (TF/IDF) is a set-based similarity measure. It considers two strings to be similar if they share distinguishing terms and is mostly used to find documents that are relevant to search keywords. The arithmetic formula is shown in equation 2.3. For example, to be considered are the three strings $x$ = (Apple Corporation, CA), $y$ = (IBM Corporation, CA), and $z$ = (Apple Corp). A traditional sequence-based measures would match $x$ with $y$ as $s(x, y)$ higher than $s(x, z)$. However, the TF/IDF

measure is able to recognize Apple as a distinguishing term, and thus would correctly match $x$ and $z$ with a higher score. The measure converts each string to a bag of terms using information retrieval (IR) terminology. For example, $x$ = aab is converted into document $B_x$ = {a, a, b}. Then TF and IDF are computed: $tf(t, d)$ refers to the number of times $t$ occurs in $d$, and $idf(t)$ refers to the total number of documents in the collection divided by the number of documents that contain $t$. The TF/IDF score between $p$ and $q$ can be computed as the cosine of the angle between these two vectors.
[DHI12]

$$s(p, q) = \frac{\sum_{t \in T} v_p(t) \cdot v_q(t)}{\sqrt{\sum_{t \in T} v_p(t)^2} \cdot \sqrt{\sum_{t \in T} v_q(t)^2}} \tag{2.3}$$

### Hybrid Similarity Measures Techniques

The hybrid similarity measures combine the advantages of the sequence-based and set-based measures. While Sequence-based techniques cover matching of misspelled or abbreviated strings, the set-based techniques cover matching strings based on shared distinguishing terms.
[DHI12]

#### *The Soft TF/IDF Measure*

The soft TF/IDF is a hybrid similarity measure. It enhances the TF/IDF measure 2.1 by softening the exact match requirement and requiring similar matching instead. The arithmetic formula is shown in equation 2.4. For example, to be considered are the tree strings $x$ = (Apple Corporation, CA), $y$ = (IBM Corporation, CA), and $z$ = (Aple Corp). The traditional TF/IDF measures would not match $x$ with $z$ because the term Apple in $x$ does not match the misspelled term Aple in $z$. The Soft TF/IDF first generate two documents $B_x$ and $B_y$ which contain terms close to each other, then computes the $s(x, y)$ as in the traditional TF/IDF measure. The measure creates two documents $B_x$ and $B_y$ as described by the TF/IDF measure. Then it computes $close(x, y, k)$ as a set of all terms in $B_x$ that have a close term in $B_y$ that satisfies $s'(t, u) \geq k$, where $k$ is a predefined threshold. $s(x, y)$ as in the traditional TF/IDF score are computed. $v_x$ and $v_y$ are the generated feature vectors and $u_*$ is the term that maximizes $s'(t, u)$ for all $u \in B_y$.
[DHI12]

$$s(x, y) = \sum_{t \in close(x,y,k)} v_x(t) \cdot v_y(u_*) \cdot s'(t, u_*) \tag{2.4}$$

## Data Matching

Similar to the string matching procedure, data matching aims to find structured data items that describe the same entities. Similar data can be joined from sources that have a different schema. For example, the tuples (Alan Mathison Turing, 555-234-5678, Maida Vale) and (Alan M. Turing, 234-5678, Maida Vale) refer to the same person. Also similar to the string matching procedure, accuracy and scalability are its main challenges. Clustering

and collective matching techniques aim to address the accuracy challenge.
[DHI12]

### Clustering Matching Techniques

*Agglomerative Hierarchical Clustering (AHC)*

The AHC is a clustering matching technique that divides data into a set of groups called clusters where members of one cluster belong together. For example, given a set of tuples $A$, AHC partitions $A$ into a set of clusters, such that all tuples in each cluster refer to an entity, tuples in different clusters refer to different entities. The algorithm begins by putting each tuple in $A$ into a single cluster. Then iteratively merges the two most similar clusters until the similarity score drops below a predefined threshold. The similarity can be computed using different methods: single link, complete link, and average link. The single link compute the similarity score as the minimal score between all tuple pairs $s(c, d) = min_{x_i \in c, y_i \in d} sim(x_i, y_j)$. The complete link maximize the computed score between all pairs of tuples $s(c, d) = max_{x_i \in c, y_i \in d} sim(x_i, y_j)$. The average link considers the total number of pairs $s(c, d) = \sum_{x_i \in c, y_i \in d} sim(x_i, y_j)/n$.
[DHI12]

### Collective Matching Techniques

*Advanced Agglomerative Hierarchical Clustering*

The advanced AHC enhances the AHC measuring procedure by taking into consideration the correlation among all tuples. The arithmetic formula is shown in equation 2.5. $sim_{attributes}(A, B)$ computes a similarity score between $A$ and $B$ based only their attributes, while $sim_{neighbors}(A, B)$ computes a similarity score between the neighborhood of $A$ and $B$. Then, the most similar clusters are merged.
[DHI12]

$$sim(A, B) = \alpha \cdot sim_{attributes}(A, B) + (1 - \alpha) \cdot sim_{neighbors}(A, B) \tag{2.5}$$

## Schema Matching and Mapping

The idea behind integration is giving the user a way to interact with the data through a single schema, also known as mediation schema. The mediated schema is a logical schema that contains the relevant information of the domain. Schema matching specifies how the elements of the source schema correspond to the mediated schema. It also indicates how to translate data across their sources and the mediated schema. For example, attribute detail in one source corresponds to attribute information in another, and name in one source is a concatenation of firstname, and lastname in another. This approach requires an understanding of the semantics of the source and mediated schemas in order to construct the mapping between them.
[DHI12]

## 2.2 Semantic-based Integration

The semantic-based integration is the process of finding the similarity between two concepts by measuring the overlap of their instance sets. The procedure is used to discover the possible similarity between a new instance and the set of instances already presented. One of its challenges lies in the fact that a schema does not adequately describe its meaning, which makes it hard for a program to process it. Furthermore, two schema elements may share the same name and yet refer to different real-world concepts. Conversely, two attributes with different names can refer to the same real-world concept. In some cases, it is hard to elaborate the matches into mapping. For example Event.localTime ≈ Event.universalTime + Location.timeZone.
[ES13, DHI12, CFMV11]

### Ontology Learning

In computer science, an ontology is an engineering artifact used to describe a real-world concept. It also consists of a vocabulary and axioms to describe their meaning. It is used to represent a conceptual model for a specific domain. The symbolic representation of knowledge can be separated from the aspects related to the application and therefore reused across different systems. This representation makes it possible for a computer to process it. Figure 2.1 shows an ontology example with a set of entities and the relation between them.
[Cim06, MS02]

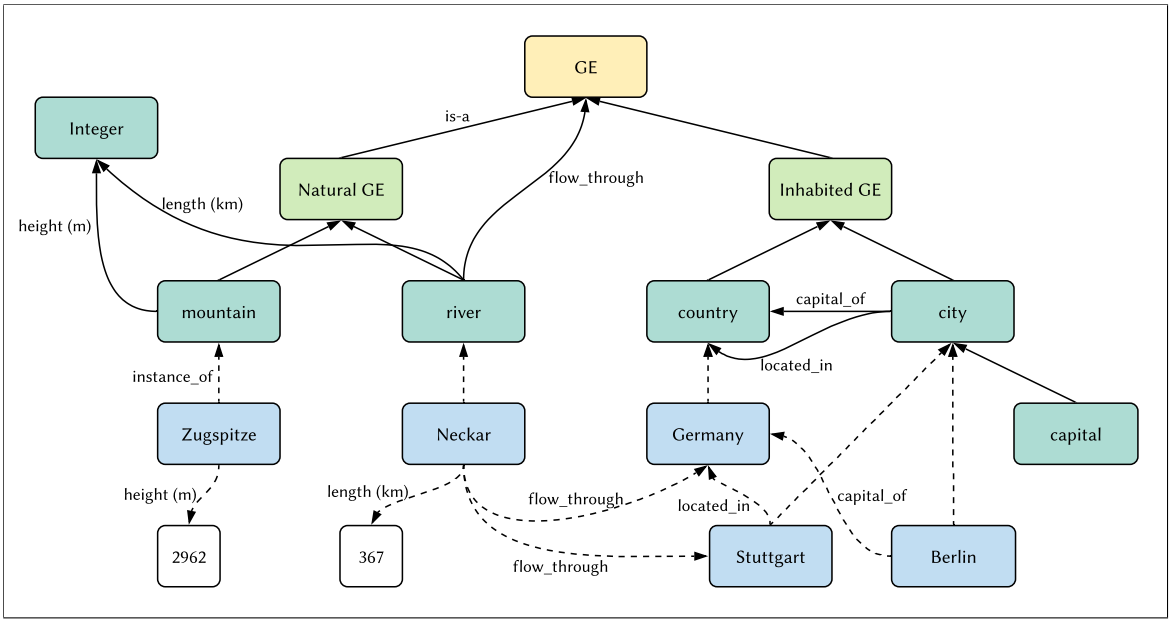


Figure 2.1: Ontology example, according to Cimiano [Cim06, p. 11]

## Ontology Alignment

The ontology alignment is the result of a matching process. It is used to identify relations between individuals of multiple ontologies. The identified correspondences are among others, one of the following: equivalence, consequence, and disjointness. This procedure is the first step toward the ontology merging and query answering procedures. Figure 2.2 shows an alignment example between two ontologies of the same domain. There are two classes of ontology matching techniques: element-level and structure-level. The element-level concludes string-based, language-based, and constrain-based techniques. The structure-level concludes graph-based, taxonomy-based, model-based, and instance-based techniques. The main challenge presented in this procedure is involving the user to validate and adjust the matching process without being lost in the design choices. Another challenge is the types of heterogeneity that might be presented in the ontologies. Syntactic heterogeneity is the case where two ontologies are not described with the same ontology language. The terminological heterogeneity which means variations in the names when referring to the same entities. Conceptual heterogeneity is the case when modeling the same domain of interest but in different ways. Semiotic heterogeneity is the case when entities are interpreted differently.
[ES13, Ehr07]

### String-based Matching Techniques

The string-based procedure belongs to the element-level matching techniques. It considers a string as a sequence of letters. The more similar the strings, the more likely they refer to the same real-world concept. The procedure is used to match names and descriptions of ontology's entities. Some examples of the string-based techniques are edit distances, and $n$-gram similarity measures.
[ES13]

#### *Token-based Distances*

The Token-based Distances belongs to the string-based matching techniques. It considers a multiset of words $s$ as a vector in which dimension is a term, and each position is the number of occurrences. After transforming the entities into vectors, metric space distances can be used to compute the similarity such as Euclidean distance, Manhattan distance, or Minkowski distance. For example, InProgress becomes In and Progress, similarity-based measures becomes similarity, based and measures. The arithmetic formula is shown in equation 2.6. $\vec{s}$ and $\vec{t}$ are the vectors corresponding to two strings $s$ and $t$ in a vector space $V$ [ES13, p. 93]. the cosine similarity is the function $\sigma_v : V \times V \longrightarrow [0\ 1]$.
[ES13]

$$\sigma_v(s,t) = \frac{\sum_{i\in|V|} \vec{s_i} \times \vec{t_i}}{\sqrt{\sum_{i\in|V|} \vec{s_i}^2 \times \sum_{i\in|V|} \vec{t_i}^2}} \tag{2.6}$$

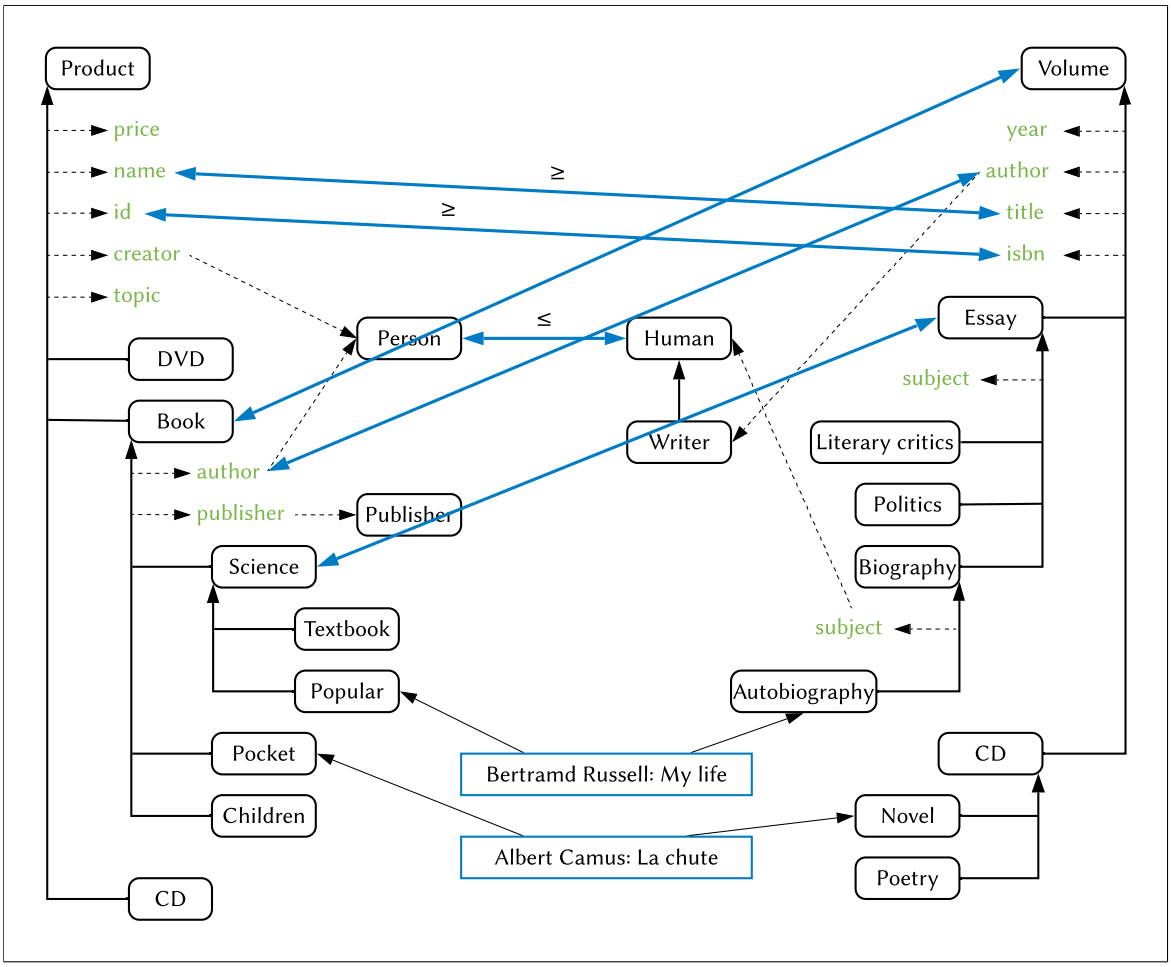


Figure 2.2: Ontology alignment, according to Euzenat and Shvaiko [ES13, p. 45]

### Constraint-based Matching Techniques

The constraint-based procedure belongs to the element-level matching techniques family. The procedure deals with the internal constraints of the defined entities, such as types, multiplicity of attributes, and keys.
[ES13]

#### *Internal Structure-based Techniques*

The internal structure-based techniques belong to the constraint-based matching techniques. Its mode of operation is to calculate a similarity measure based on the structure of entities, in addition to comparing the names and identifiers. The structure of entities considering their properties (names, keys, data types, domains, cardinalities), their relations, and their multiplicity. The procedure is combined with element-level techniques to reduce the number of candidate correspondences and eliminate incompatible properties. Figure 2.3 shows and example for an internal structure comparison technique, considering

the properties and cardinality associated with the Product and Volume. The procedure's limitation is caused by the fact that the internal structure does not provide much information to compare, as many different objects can have the same properties with the same data types.
[ES13]

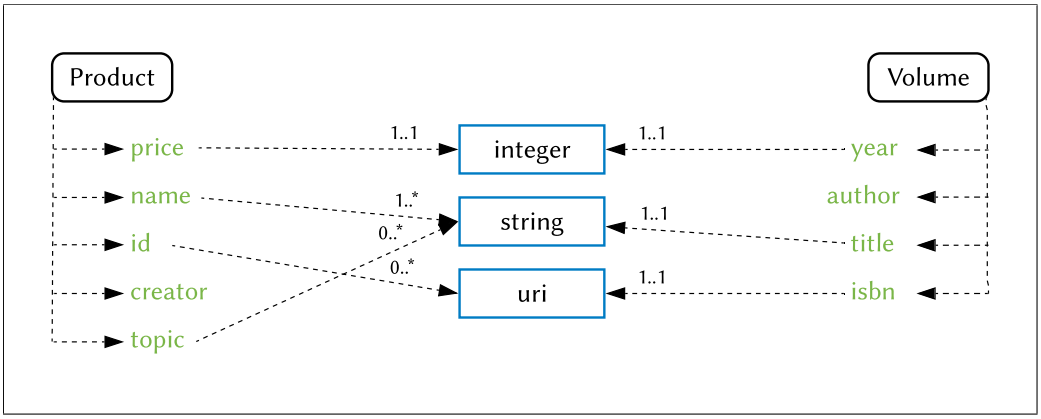


Figure 2.3: Internal structure comparison, according to Euzenat and Shvaiko [ES13, p. 107]

### Graph-based Matching Techniques

The graph-based procedure belongs to the structure-level matching techniques. It considers the input ontologies as a graph whose edges are labeled by relation names. In case two nodes are similar, their neighbors must also be similar. The procedure finds correspondences between elements by finding a common homomorphic subgraph. Graph homomorphisms are mappings of the vertices while preserving adjacency. For example, given $G$ and $H$ two graphs, homomorphisms of $G$ to $H$ are paths in $G$ mapped in $H$ without increasing the distances. The example 2.7 represents a relation-based example on the subClasses of Book from the ontology presented in figure 2.2.
[ES13, HN04]

$$\begin{aligned} subclass(\text{Book}) = subclass^{-}(\text{Book}) &= \{\text{Science, Pocket, Children}\} \\ subclass^{+}(\text{Book}) &= \{\text{Science, Pocket, Textbook, Popular, Children}\} \\ subclass^{-1}(\text{Book}) &= \{\text{Product}\} \\ subclass^{\uparrow}(\text{Book}) &= \{\text{Textbook, Popular, Pocket, Children}\} \end{aligned} \tag{2.7}$$

### Instance-based Matching Techniques

The instance-based procedure belongs to the structure-level matching techniques family. The mode of its operation is to compare sets of instances of classes to determine whether these classes match or not. For example, $A$ and $B$ are two classes with one of the possible relationships. Equal ($A \cap B = A = B$), contains ($A \cap B = A$), contained-in ($A \cap B = B$), disjoint

($A \cap B = \emptyset$). The matching is facilitated when two ontologies share the same set of instances. However, in the case of incorrect data, the system may deliver wrong correspondences. To overcome this limitation, Hamming distance can be used to calculate the corresponds based on the size of symmetric differences normalized by the size of the union. The arithmetic formula is shown in equation 2.8. Hamming distance can still produce a short distance even in case of misclassified instances due to the usage of symmetric normalization.
[ES13]

$$\sigma(x, y) = \frac{|x \cup y - x \cap y|}{|x \cup y|} \tag{2.8}$$

## Ontology Merging

The ontology merging procedure is used to integrate matched ontologies $o$ and $o'$ into a new ontology $o''$ based on a computed alignment. For example, consider the following alignment results: (id = isbn) and (book ≤ Volume). The resulting ontology will have the following individuals: (o:id owl:equivalentProperty o:isbn) and (o:Book rdfs:subClassOf o:Volume). The procedure does not require a total alignment between the individuals of both ontologies. Entities with no correspondences will remain unchanged. Figure 2.4 shows the internal functionality of the ontology merging procedure.
[ES13]

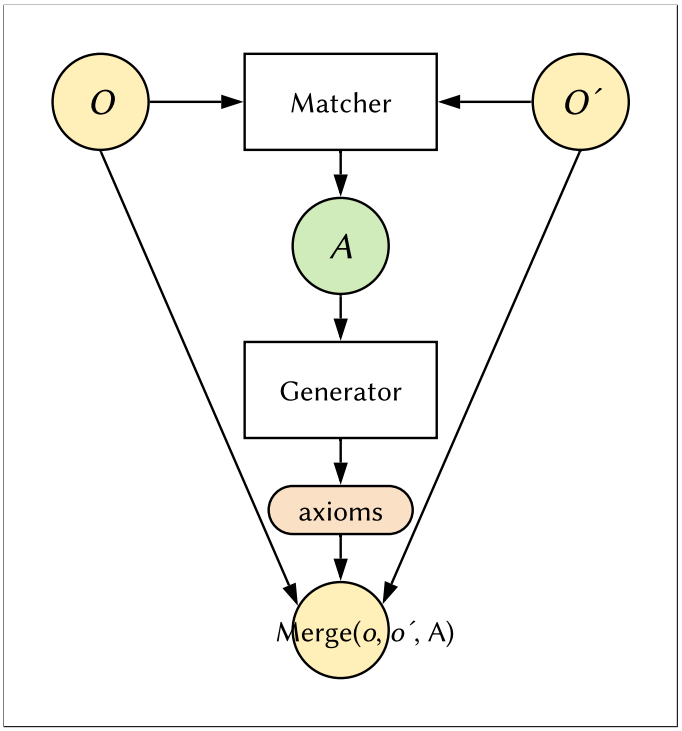


Figure 2.4: Ontology merging, according to Euzenat and Shvaiko [ES13, p. 379]

# 3 Related Work

There has been a range of work conducted to address the syntactic- and semantic-based integration approaches. Each work focused on identifying the gaps and limitations in each approach and testing the suggested methodology on a practical example. The research conducted by Fadi Jabbour focused on the syntactic-level by using string matching techniques to achieve a semi-automated model transformation. Yong Wang et al. focused on the semantic-level by using ontologies to achieve an automatic adaptation of component interfaces.

This chapter reviews both research and the approaches to tackle the problems encountered in their work, which will help to identify the gaps that make it possible to join both approaches.

## 3.1 Model Transformation for Domain Specific Architecture Languages in the Automotive Software Development

The research work of Fadi Jabbour emerged to address the increase in the complexity of the software systems. It is the result of the growing number of software systems that need to interact together. Using a standardized architecture description languages is the first step towards achieving interoperability among integration tools, and therefore, approach this problem. The fact that software components are developed by different parties leads to a different representation of the models even in case of modeling the same domain. As for an example, Architecture Description Languages (ADLs) is a description language that belongs to the same application domain and shares the same semantics; however, architectures use different notations for describing the same software architecture.

The research work contributed to finding a solution for achieving interoperability between different ADLs by developing an automated transformation mechanism that gives the user the ability to evaluate the result obtained by the transformation.

The work encountered integration gaps between ADLs, which can be categorized into three different levels, ordered from the least to the most abstracted. First, the technology level, which refers to the technologies used to build the ADLs, such as Extensible Markup Language (XML), and XML Metadata Interchange (XMI). Second, the modeling language level, which refers to the existence of different syntaxes. Third, the structure level, which refers to the existence of different characteristics of the model. The integration tool needed to address and solve those challenges by implementing a tool that is compatible with the

commonly used technologies and using a well-defined mapping between the model structures such as the weaving model and model transformation.

Figure 3.1 shows the mode of operation for the proposed model weaving as part of the model transformation concept. The model weaving produces a mapping between the source and target metamodel, which can be used for transforming the model using the model matching procedure.

The study introduced a system with two components: the reusable transformation rules, and the improved model weaving operation. The idea behind the reusable transformation rules is to define transformation rules to help to transform and therefore match the source and target models. The idea behind the improved model weaving operation is to give the user the ability to create relations between source and target metamodels. The overall approach can be further improved by applying the weaving model on smaller logical parts by matching sub-parts of the source and target metamodels, which can be combined to create the overall model transformation.

The matching algorithm, which generates the weaving model, operates on all elements of the source and target metamodels (e.g., classes, attributes, references). Using all structure information from the model is a well-known approach, as presented in chapter 2. The developer can update and correct the weaving result.

While the primary goal of the study was to provide an automatic approach for the model transformation, but the results of the model weaving were not satisfying, since the element names are abbreviations that consist of the first letter of each word. In many cases, the attributes share the same name “value” and type “string” with no size limit. Therefore, the system is not applicable in a real industrial environment, because errors that occur in the early stages of the algorithm will propagate in later stages as well. Thus, the effort needed to adjust the results is more significant than constructing the weaving model from scratch.

Therefore, the idea to obtain an automatic approach was replaced by a semi-automatic multi-steps algorithm that requires the developer’s interaction, which resulted in correcting errors at each stage of the algorithm and preventing their propagation. The algorithm starts by matching classes of source and target metamodels based on their internal structure such as name, size, multiplicity, and similarity of attributes. The developer then validates and corrects the results. The algorithm then searches for equivalent properties such as name, type, and lower and upper bound in the equivalent classes found in the previous matching step. The developer once again validates and corrects the results.

The work was concluded by providing future research suggestions, for example, achieving bidirectional transformation. The research investigated the creation of transformation only from source to target models. However, It would be beneficial to construct a bidirectional transformation mechanism, also from the target back to the source model.
[Jab15]

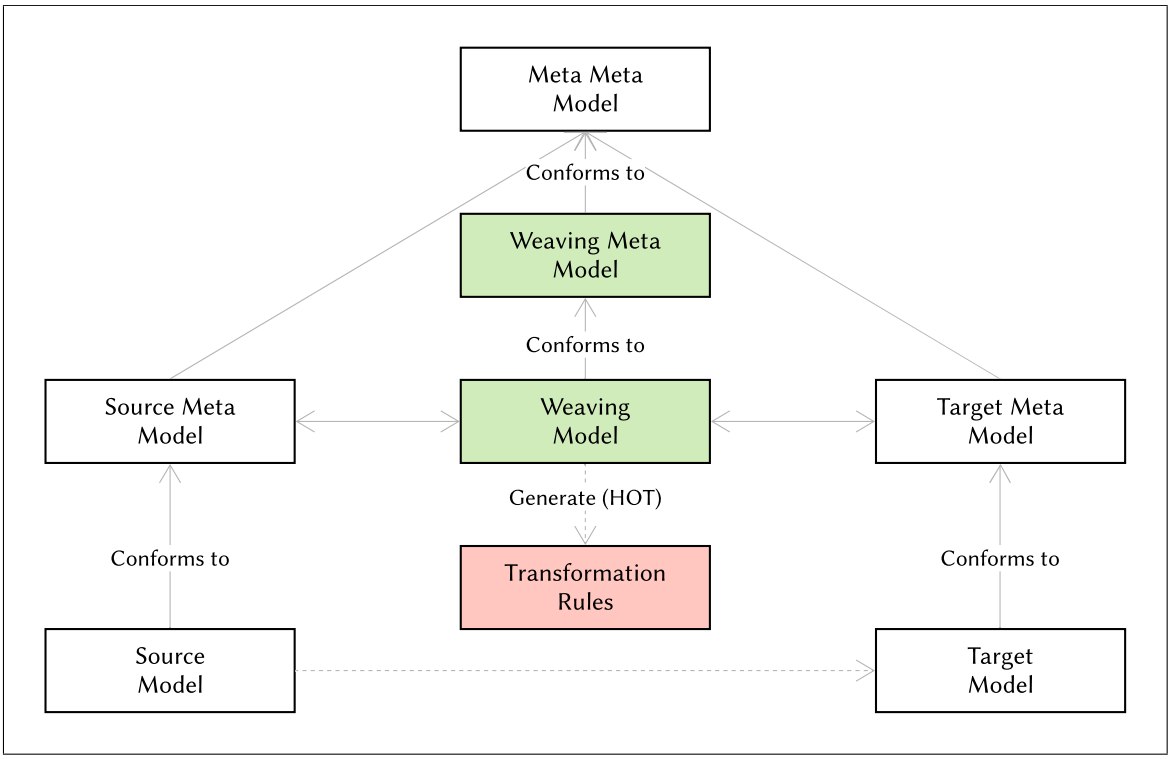


Figure 3.1: Model weaving, according to Jabbour [Jab15, p. 21]

## 3.2 Ontology-based Automatic Adaptation of Component Interfaces in Dynamic Adaptive Systems

The research work of Yong Wang et al. emerged to address the reusability of components in a new application context using semantic models. The research reviewed dynamic adaptive systems that change their behavior at runtime based on changes in the users' requirements or system's environment. Dynamic Adaptive System Infrastructure (DAiSI) is an example of a component-based dynamic adaptive system. DAiSI's components are defined using an interface that describes required and provided services.

The research work contributed to finding a solution for reusing components in application contexts outside the scope of their intended implementation. The goal is to couple provided and required services that are semantically compatible but syntactically not. Therefore, three types of incompatibility are covered: different naming, different data structure, and different control structure. The three types of incompatibility are shown in figure 3.2 The different naming is the case where names of interfaces or the functions do not match but share the same semantics. The different data structure is the case where the parameters differ in their data types; however, the encapsulated data can be mapped to each other. The different control structure is the case where different interfaces provide information which can be composed rather than mapped.

The study introduced the usage of a central ontology as the common knowledge base, which provides the mapping between the semantically compatible provided and required services. The approach extends the DAiSI components' interfaces through developing an ontology-based adapter as shown in figure 3.3 The central ontology consists of three layers: upper layer, application layer, and interface layer. The upper layer's ontology is called UpperOntology, which contains basic knowledge definition. The application layer's ontology contains all definitions that are relevant for an application. The interface layer represents the domain interfaces with their names, methods, parameters, and data types.

Figure 3.4 shows the mode of operation for the proposed system. First, the information collector transforms the annotations in the interface definition. Then, a mapper search in all instances of the ontology to find a mapping between required interfaces that can be used by provided interfaces. Finally, the adapter component is created to link the mapped interfaces regarding one of the incompatibilities presented earlier.

The proposed approach has the advantage of making every part of the ontologies and interfaces being developed separately. Thus, opening further research in the field of ontology mapping and merging so that layers can be merged with other dynamic adaptive system domains.

The work was concluded by providing further research suggestions, for example, the usage of distributed ontologies, so that every component is directly linked to an ontology which describes its structure.
[WHSR16]

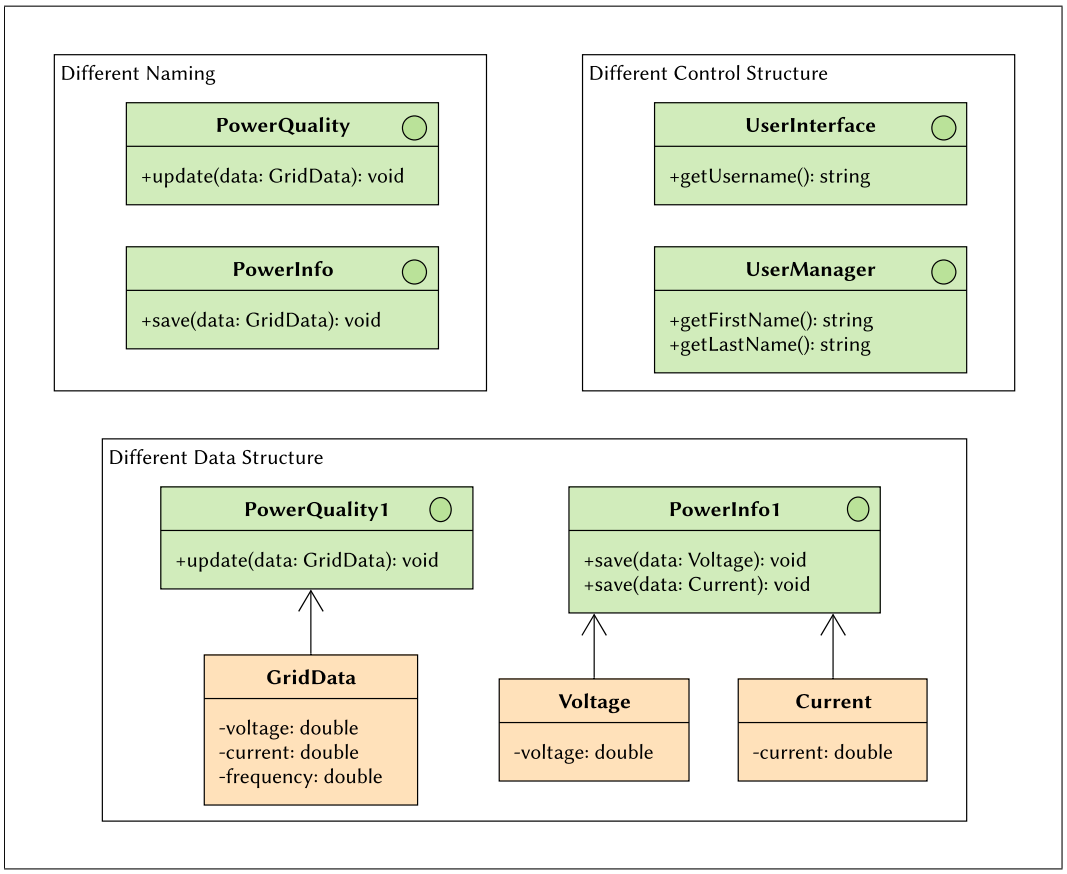


Figure 3.2: Taxonomy of incompatibilities between interfaces, according to Wang et al. [WHSR16, p. 2]

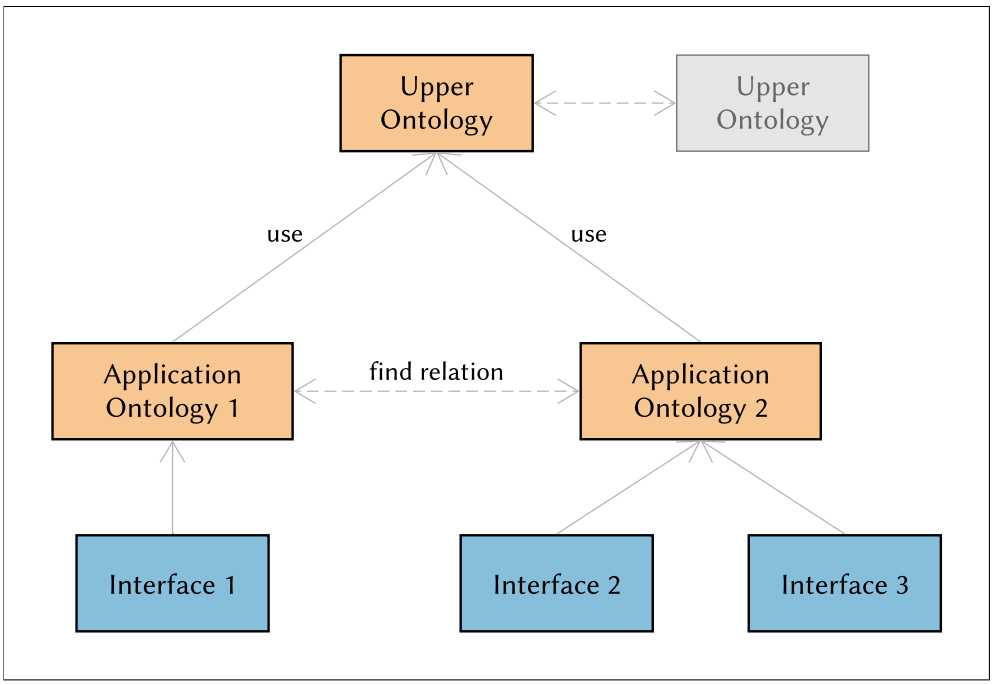


Figure 3.3: Three-layer ontology structure, according to Wang et al. [WHSR16, p. 5]

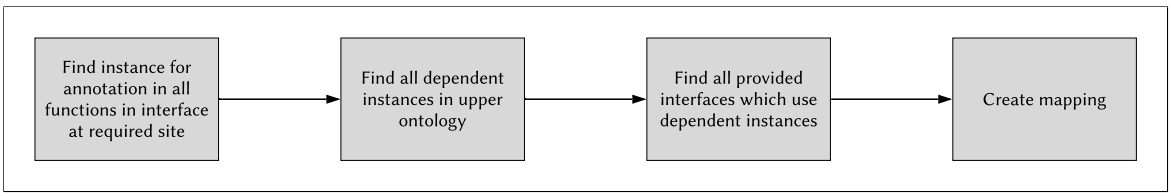


Figure 3.4: Process for mapping required interfaces to provided interfaces, according to Wang et al. [WHSR16, p. 6]

# 4 Approach

As stated before, the goal of this work is to introduce a domain knowledge life cycle and a toolchain to help the creation and development of domain representations on semantic bases. The life cycle consists of four procedures: composition, transformation, matching, and merging. The following chapter outlines each procedure. Then compare existing approaches and tools with the help of evaluation matrices, to compose the toolchain.

## 4.1 Ontology Composition Procedure

An Ontology is a specification of an abstract and simplified view of a particular topic. It consists of a set of vocabularies and axioms to describe a specific domain. OWL 2 is an ontology language for the semantic web. The structure of the OWL 2 language, as well as its components and their relations, are described in figures 4.1 and 4.2. The following subsections describe each component of the OWL 2 language.
**Internationalized Resource Identifier (IRI)** is an internet protocol standard that extends the Uniform Resource Identifier (URI) by using the universal character set, instead of the American Standard Code for Information Interchange (ASCII). IRIs are used to identify ontologies and their versions. Additionally, the elements of an ontology may also be identified using IRIs.
**Annotations** are used to associate information with an ontology, e.g., description field or current version.

```
owl:backwardCompatibleWith( <http://www.w3.org/2019/Example> )
```

**Axioms** are the main component of an ontology and used to describe truth inside the domain. They can be declarations or about classes, objects, or data.
**Classes** describe sets of individuals. For example, the classes a:Teacher and a:Person can be used to represent the set of all teachers and persons, respectively.

```
SubClassOf( a:Teacher a:Person )
```
Each teacher is a person

**Datatypes** are the same as classes, with only the difference that they refer to sets of data values instead of individuals. For example, the datatype xsd:integer denotes the set of all integers and can be used as axioms.

```
DataPropertyRange( a:hasAge xsd:integer )
```
The range of the a:hasAge data property is xsd:integer

**Object properties** connect pairs of individuals. For example, the object property a:teachesSubject can be used to present the parenthood relationship between individuals.

```
ObjectPropertyAssertion( a:teachesSubject a:Alan a:CS101 )
```
Alan teaches CS101 subject

**Data properties** are the same as object properties, with only the difference that they connect individuals to literals instead of to one another. For example, the data property a:hasName can be used to associate a name to a class.

```
DataPropertyAssertion( a:hasName a:Alan "Alan Turing" )
```
Alan's name is "Alan Turing"

**Annotation properties** provide annotations to an ontology, axiom, or an IRI. For example, the annotation property rdfs:comment displays additional information.

```
AnnotationAssertion( rdfs:comment a:Alan "Alan was a mathematician" )
```
This axiom provides a comment for a:Alan

**Individuals** represent objects from a domain. Individuals are given an explicit name that is used to refer to the same object. Anonymous individuals can not be accessed globally and are thus local to the ontology containing its instantiation.

```
ClassAssertion( a:Teacher a:Alan )
```
Alan is a teacher

**Literals** represent data values such as string, decimal, or integer. It consists of a lexical form denoting its value and a datatype.

```
"1"^^xsd:integer
```
A literal that represents the integer 1

[AB13, SCH+12, MPB+12, GDD09, Hen01, Gru93]

## 4.2 Data Source Transformation Procedure

As described in section 2.2, the ontology learning process is the first component in a semantic-based integration process. In this context, we describe it as a data source transformation process. The procedure extracts the vocabulary and data which describes the domain. This vocabulary may be presented in a schema description language as in XML Schema Definition (XSD) or a tabular header as in comma-separated values (CSV). The extracted vocabulary and data are then inserted into a knowledge base. It is beneficial to separate knowledge representation from procedural aspects related to its application in order to reuse the knowledge across other systems.

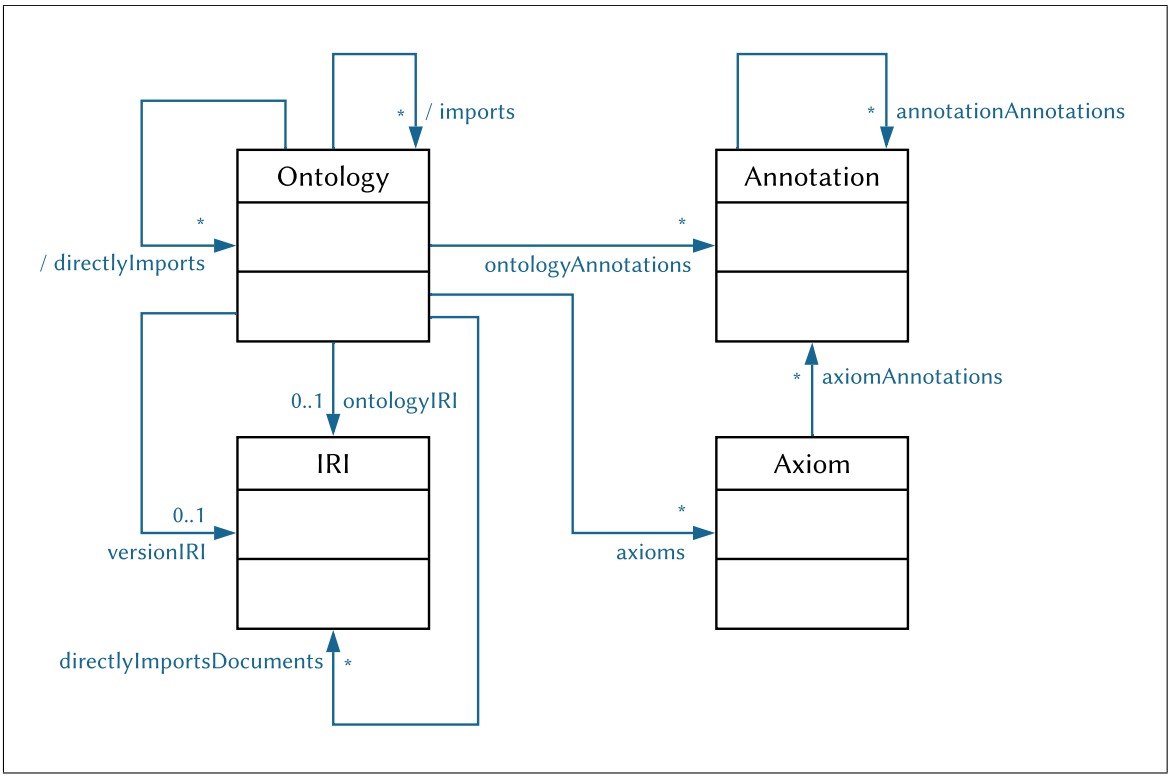


Figure 4.1: Structure of OWL 2 ontologies, according to Motik et al. [MPB+12, p. 8]

Despite its benefits, the procedure also encounters some limitations. First, the extracted vocabulary is based on the assumptions of the domain expert constructing it. Hence, the presence of the domain expert is required to approve the validation of the extracted knowledge. Second, the construction of ontologies is costly due to the trade-off between a large amount of knowledge, and a provision of abstraction to enhance its reusability. Therefore, the correctness and consistency can not be guaranteed.

The traditional ontology learning builds on natural language processing and machine learning, using methods such as named entity recognition, anaphora resolution, and morphology components. The named entity recognition method extracts a small number of classes from domain-specific collections of unstructured texts. In the scope of this study, such techniques are not required, as the extraction is based on structured data sources. It is sufficient to transform the structured data into a knowledge representation.
[Cim06]

## 4.3 Ontology Matching Procedure

As described in section 2.2, the ontology matching process is the second component in a semantic-based integration process. It describes the process of finding correspondences between entities of different ontologies of the same domain. Alignment is the output of a matching processes and is used for the evolution of ontologies. The matched ontologies

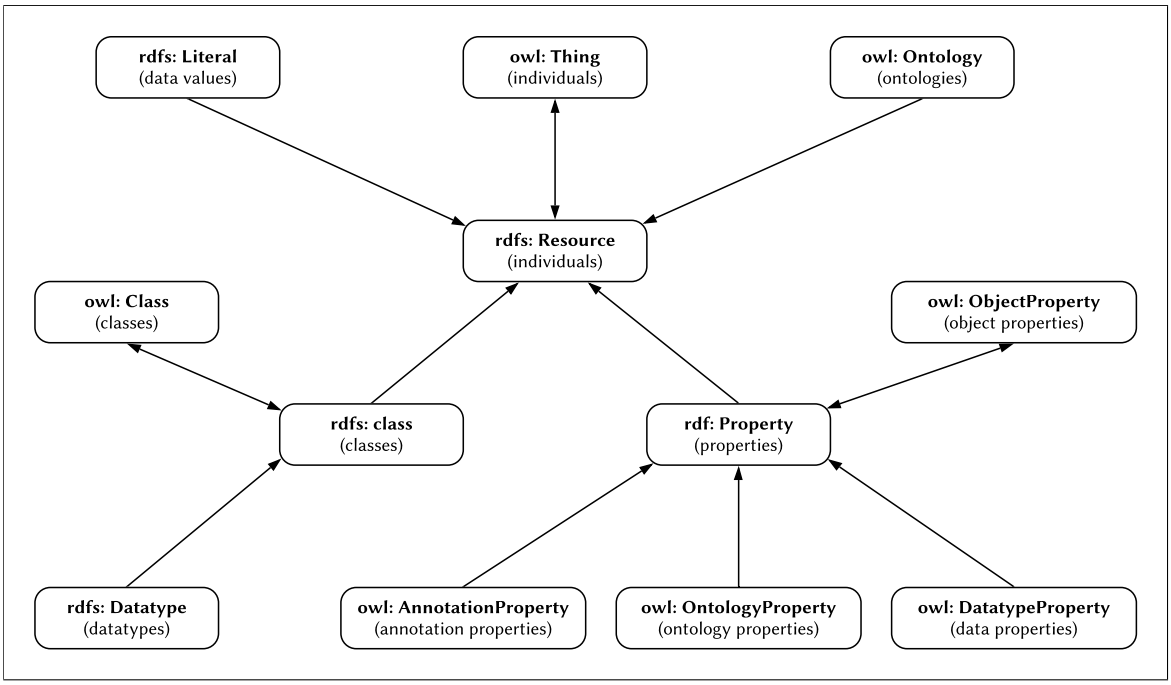


Figure 4.2: Parts hierarchy of the OWL 2 RDF-based semantics, according to Schneider et al. [SCH+12, p. 6]

enable the entities to interoperate.

Euzenat and Shvaiko [ES13] defined the ontology alignment life cycle. As shown in figure 4.3, the life cycle workflow is subdivided into the following steps. The alignments are first generated using a matching process. Then the alignment result can go through an iterative evaluation and enhancement loop if necessary. Finally the alignment can be used for procedures like query answering or merging. The proposed alignment life cycle may be performed manually or automatically.

The ontology matching procedure presents one of the solutions to the semantic heterogeneity problems described in section 2.2. The alignment produced from the matching can be used in various tasks such as ontology merging, query answering, and data translation. Moreover, the same procedure can be applied to database schema matching.

Despite its benefits, the procedure also encounters some limitations, which are the types of mismatches. When matching different ontologies, mismatches might occur on two different levels: language-level and ontology-level. The Language-level mismatch, which describes the differences in the expressiveness of ontology languages. The expressiveness is the presence of disjoints, negations, expression, unions, intersections, etc. The ontology-level mismatch, which describes the differences in the structure or semantic of ontologies.

Euzenat and Shvaiko [ES13] reviewed about 100 ontology matching systems. The following can be observed: 50% of the systems are schema-based, 25% are mixed, i.e., rely on the schema and instances. Moreover, most of the systems operate on two input ontologies from the same domain and handle tree-like structures by focusing on the discovery of one-to-one correspondences by computing similarity measures in the [0 1] range.

[ES13]

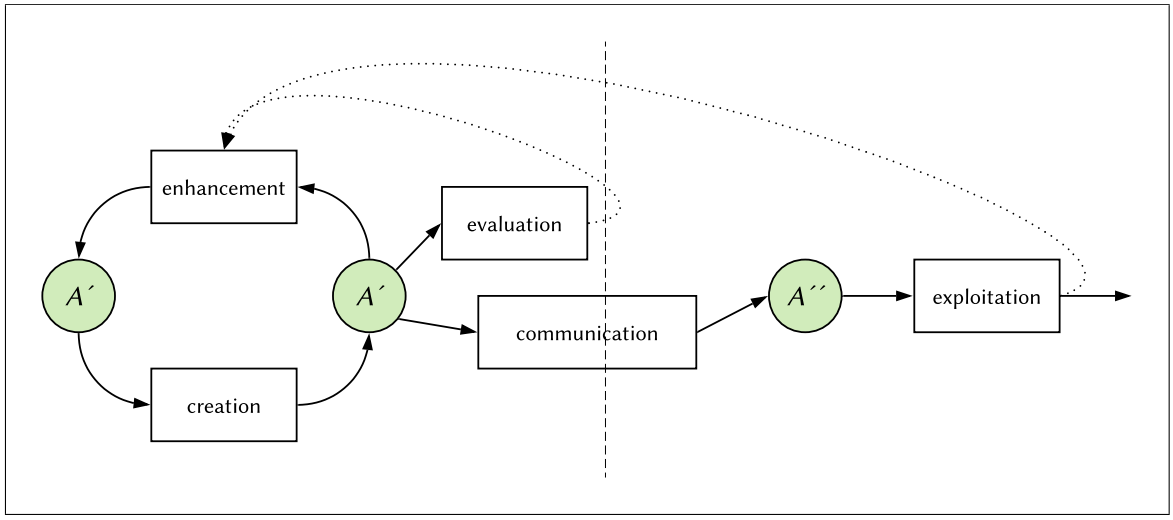


Figure 4.3: Ontology alignment life cycle, according to Euzenat and Shvaiko [ES13, p. 57]

## 4.4 Ontology Merging Procedure

As described in 2.2, the ontology merging process is the last component of a semantic-based integration process. It describes the process of obtaining new ontology from two matched ontologies based on their alignment. However, a total alignment is not required as entities with no correspondences will remain unchanged in the merged ontology. The process is used for different purposes, such as connecting ontologies to a common upper-level ontology.

There are two types of merging systems. The first uses the alignment result as bridge rules for the entities and creates the merged ontology accordingly. The other system creates an ontology mapper in query forms in order to answer the queries from the merged global ontology.
[ES13]

## 4.5 The Ontology ToolChain

One of the main goals of this study is to define a domain knowledge development life cycle and a toolchain to help by the creation and development of semantic domain representation. The life cycle is used in the evolution of semantic representations, e.g., ontologies, for domains of interest. Thus, a traditional evolution of an ontology would include its creation and potential enhancement through the merging process with other ontologies of the same domain. As mentioned in section 4.4, the merging procedure requires a partial or entire alignment between entities of different ontologies of the same domain. Therefore, the evolution of an ontology would also include a matching process to find correspondences

with other ontologies of the same domain. The last process in the life cycle is the transformation. As mentioned in section 2.2, the most efficient matching techniques are the constraint- and instance-based, which take into consideration the structure and instances of the entities in the ontologies which need to be matched. Thus, the data source transformation process is essential to append data structure properties and instances from the data source to its semantic representation (ontology).

As shown in figure 4.4, the toolchain consists of four tools: composition, transformation, matching, and merging. Each tool is used to achieve one functionality of the toolchain mentioned above and can be executed independently. The toolchain delivers an integrated domain knowledge, which is the alignment result of the transformed knowledge from the data sources of some domain of interest. One essential characteristic of the toolchain is the presence of the feedback mechanism. In other words, the user is able to confirm or adjust the output at every stage.

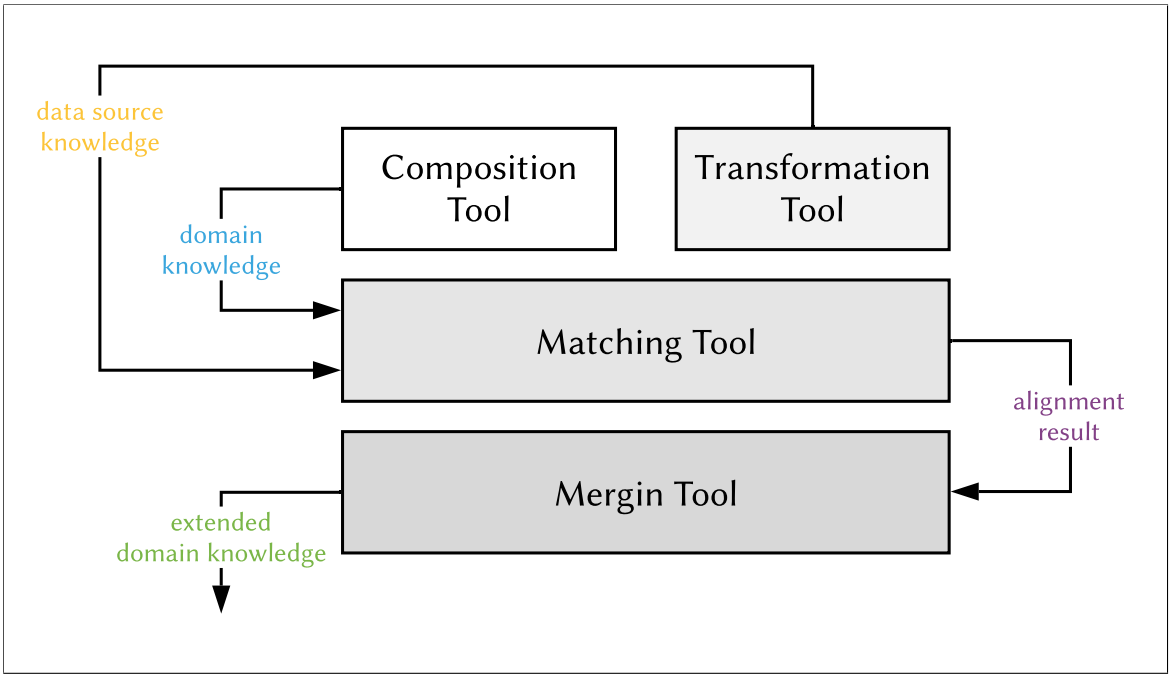


Figure 4.4: The ontology toolchain structure

## Composition Tool

The composition tool is responsible for the creation of an ontology for a specific domain of interest. Using the composition tool, a user can create a domain knowledge base for a particular application. The tool is capable of applying the ontology composition procedure presented in section 4.1. Figure 4.5 depict the structure of the tool. Some of the popular ontologies construction tools in the market are Protégé, WebOnto, OilEd, SWOOP, OntoSaurus, Ontolingua Server, WebODE, and OntoEdit. Protégé and SWOOP were selected for the evaluation conducted in chapter 5.

Protégé is a popular, well-known open-source ontology construction tool developed at Stanford University. It provides a graphical user interface (GUI) for the creation, visualization, and adjustment of ontologies, and supports different ontology language formats such as RDF/XML, OWL/XML.

SemanticWeb Ontology Overview and Perusal (SWOOP) is a simple OWL browser and editor for ontologies developed at the University of Maryland.
[AB13]

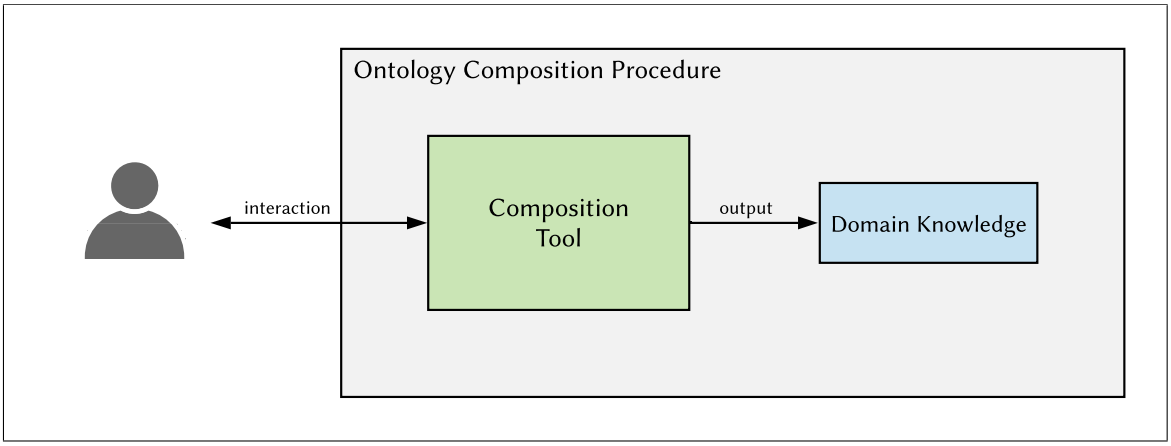


Figure 4.5: The composition tool structure

## Transformation Tool

The transformation tool is responsible for adding the structure properties and data instances to an ontology. Using the transformation tool, a user is able to construct an ontology that semantically describes a data source. The user is also able to adjust the generated ontology to apply axioms and semantic relations that are not expressed in the schema of the data source. The tool is capable of applying the data source transformation procedure presented in section 4.2. Figure 4.6 depict the structure of the tool. Protégé and Tarql were selected for the evaluation conducted in chapter 5.

Protégé contains a built-in functionality for generating instances, annotations, and axioms from spreadsheet documents (e.g., .xlsx and .xls).

Tarql is a commant line interface, used for converting CSV files to Resource Description Framework Schema (RDFS) using SPARQL Protocol and RDF Query Language (SPARQL) syntax. SPARQL is a Structured Query Language (SQL)-like query language for querying RDF-based documents. It is an accepted standard, however, not intended for ontology representation, but rather for querying. It can be used to extract information from RDF documents, and construct new ones by matching filter patterns against the target graph of the query.
[Pau11, GDD09]

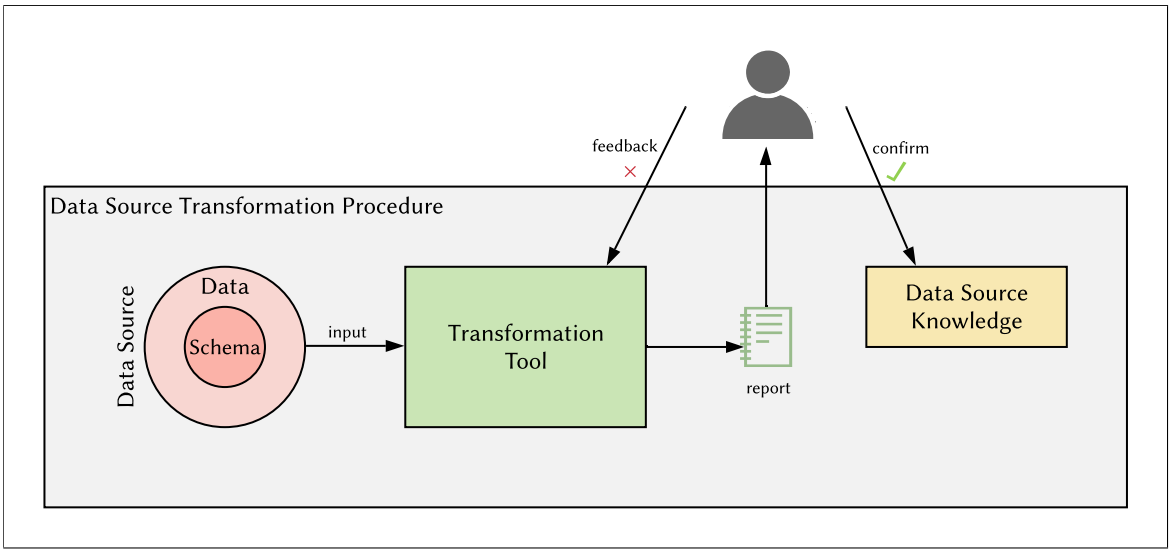


Figure 4.6: The transformation tool structure

## Matching Tool

The matching tool is responsible for finding alignment between entities of the constructed ontologies: the ontology representing the domain and the ontology generated from the data source. Using the matching tool, a user is able to find elements of two ontologies, which refer to the same concept. The user is also able to confirm or adjust the resulted alignment in case of wrong correspondences. The tool is capable of applying the ontology matching procedure presented in section 4.3. Figure 4.7 depict the structure of the tool. COMA++ and AML were selected for the evaluation conducted in chapter 5.

COMA is a schema matching tool contains six elementary, and five hybrid matching algorithms that implement string-based matching techniques such as $n$-gram and edit distance. The ontologies are imported and encoded as a directed acyclic graph (DAG), where the elements define the paths in the graph. COMA++ is an enhanced version of COMA that provides fragment-based matching and a graphical user interface (GUI).

AgreementMakerLight (AML) is a simple version of the AgreementMaker (AM) software, which offers a wide range of matching algorithms and a graphical user interface. The software is designed to handle large-scale ontologies and is capable of finding 1:1, 1:$m$, $n$:1, $n$:$m$ alignments between the entities. The matching process is subdivided into two modules: similarity computation and alignment selection. The software offers an interface to adjust the evaluation strategies and, therefore, the matching algorithm. [ES13]

## Merging Tool

The merging tool is responsible for merging a set of alignment results with the constructed domain ontology. Using the merging tool, a user is able to extend the constructed domain ontology with the interconnections found within the semantic representation of the data

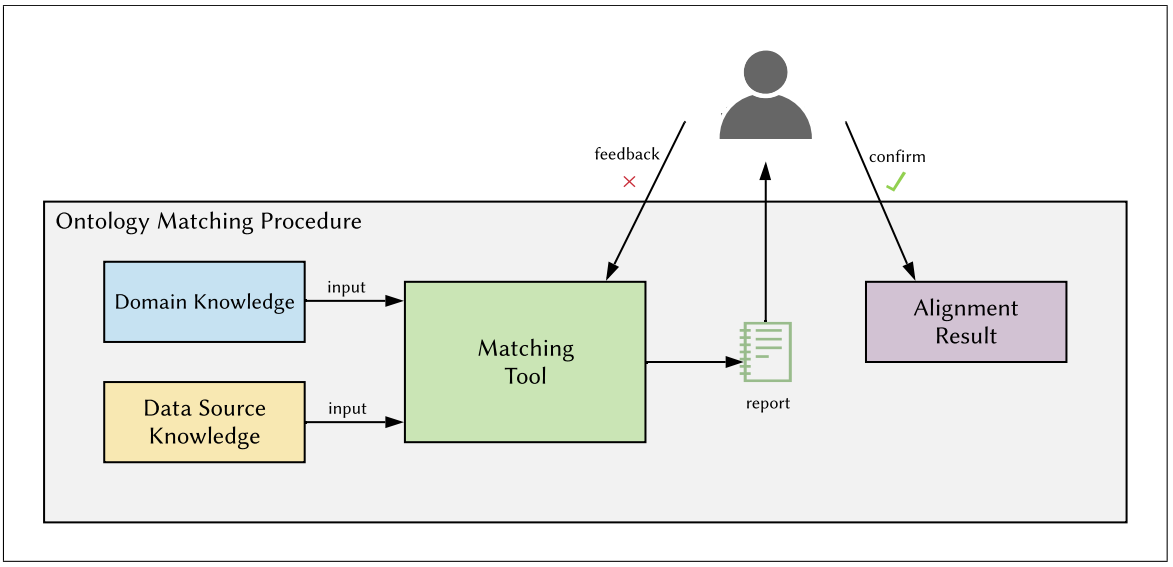


Figure 4.7: The matching tool structure

sources. The tool is capable of applying the ontology merging procedure presented in section 4.4. Figure 4.8 depict the structure of the tool. Protégé and OntoMerge were selected for the evaluation conducted in chapter 5.

Protégé contains a built-in functionality for merging ontologies, whether by extending an existing ontology or creating a new one. It is also capable of merging alignment results encoded in an ontology language format such as RDF/XML, OWL/XML.

OntoMerge is a system for ontology merging and translation. The merging procedure is conducted by taking the union of the axioms defined in both ontologies. Bridging axioms and rules are provided by a domain expert or matching algorithm to relate the terms in both ontologies. The translation task follows the merging procedure, which generates ontology extensions and query answering for both ontologies.
[ES13]

## 4.6 The ToolChain Criteria

The next step in the approach is to evaluate the proposed life cycle and tools. For this task, a set of mandatory criteria is assigned in order to be investigated and evaluated. Those criteria are: time-consuming, quality, validation, and post-processing. The reason behind choosing those criteria lies in their interrelation. In other words, the score assigned to one criterion might have an impact on the others. Moreover, investigating those criteria will help in answering the research questions in section 1.3. Further criteria can also be investigated in future research works.

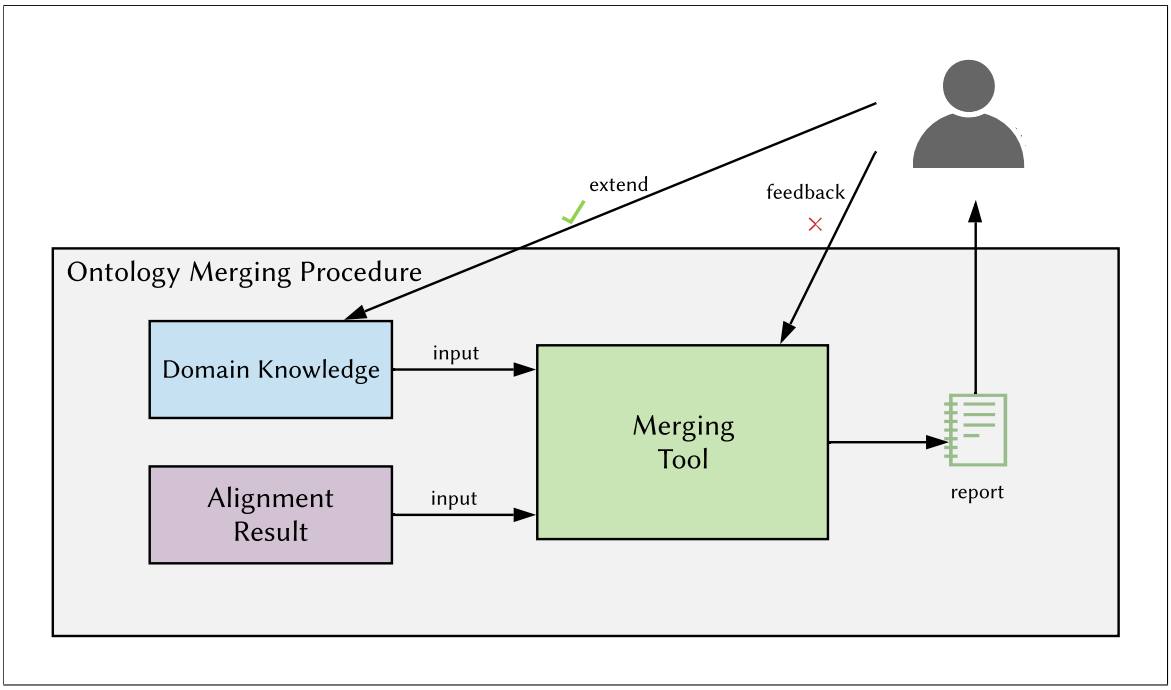


Figure 4.8: The merging tool structure

## Time-Consuming

The time-consuming criteria aim to evaluate the time spent during the interaction with the tool to get the first output regardless of the reiteration process, which is responsible for optimize the output. The post-processing criterion attends to evaluate the reiteration process. This criterion is rated by one of the following values: low, medium, and high. Low is the optimal value, which indicates that a user with limited expertise is able to interact with the tool. In other words, the tool is easily manageable and does not require any substantial experience.

## Quality

The quality criteria aim to evaluate the anomaly between the actual and expected output from the tool regardless of the reasoning process, which is responsible for checking conceptual mistakes. This criterion is rated by one of the following values: low, medium, and high. High is the optimal value, which indicates that the output satisfies the user's expectation.

## Validation

The quality criteria aim to evaluate the correctness of the output, also known as reasoning. Reasoning on ontologies is used to discover conceptual mistakes. This criterion is rated by one of the following values: low, medium, and high. High is the optimal value, which

indicates that the output is semantically correct.
[Pau11]

### Post-Processing

The quality criteria aim to evaluate the complexity of the reiteration process to optimize the output if needed. This criterion is rated by one of the following values: low, medium, and high. Low is the optimal values, which indicates that no reiteration is needed, and no conceptual mistakes are found in the result.

## 4.7 Evaluation Matrix

The evaluation matrix approach is used for the evaluation of proposed tools against the specified criteria. Table 4.1 represents an example for the evaluation matrix. Since one of the objectives of this thesis is to investigate the gap between domain experts and ontology engineers, both roles are evaluated separately. The evaluation's results are going to help in identifying the advantages and limitations of the domain engineering approach. Moreover, the deduced results are also going to help to answer the research question in section 1.3.

Based on the conducted literature and related work review in chapter 2 and 3, the following assumptions are expected. A sub-optimal score of one criterion will propagate and affect the score of at least one other criterion. Furthermore, based on the evaluated user's role, some criteria will have an optimal score as a result of the user's expertise.

| | Tool A | Tool B |
|---|---|---|
| Time-Consuming | | |
| Quality | | |
| Validation | | |
| Post-Processing | | |

Table 4.1: Evaluation matrix example

# 5 Evaluation

In chapter 4, a toolchain was introduced to help in the creation and development of semantic representation for domains of interest. As mentioned before, we aim to evaluate the toolchain as used from a domain expert and ontology engineer separately, hence, identify and reduce the gap between both roles. The result of the evaluation is as follows.

1. The ontology engineer achieved a high score in the quality criterion; however, a low score in validation.

2. The domain expert achieved a high score in the validation criterion; however, a low score in quality.

3. For all evaluation processes, a low score in the post-processing criterion is obtained.

The rest of this chapter explains in detail the evaluation of each tool.

## 5.1 Composition Tool

One alternative to constructing an ontology from scratch is to import the semantic representation of the domain of interest from a knowledge base such as wikidata. We dropped this concept due to the complexity found in the wikidata knowledge base. figures 5.1, 5.2, and 5.3 show an ontology example for the weather domain with more than 50 classes and 20,000 axioms. We argue that the effort needed to adjust the imported ontology is more significant than constructing the ontology from scratch. In the following two sub-sections, the evaluation result of the domain expert and ontology engineer is discussed.

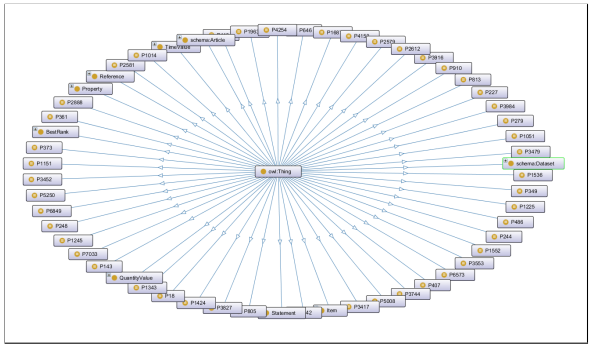

Figure 5.1: The weather ontology by wikidata

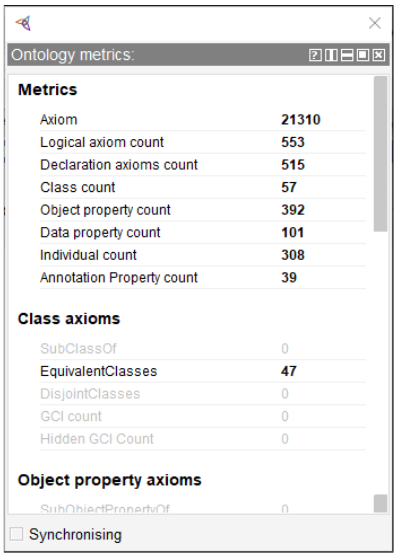


Figure 5.2: The weather ontology metrics by wikidata

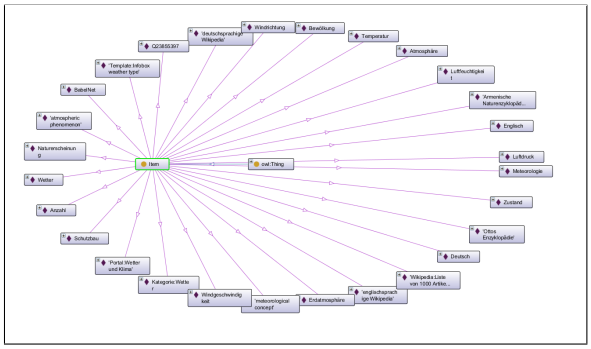

Figure 5.3: Sub-elements of the weather ontology by wikidata

### Ontology Expertise

Table 5.1 represents the resulted evaluation matrix. both scores for time-consuming and quality are optimal, because the ontology structure knowledge is presence as a part of the ontology expertise. However, the score for validation is sub-optimal, because the domain knowledge is not present and, therefore, an ontology engineer is unable to validate the constructed ontology based on the interest of the domain. We believe that the presence of the domain knowledge will increase the validation criterion score and, therefore, decrease the post-processing score.

### Domain Expertise

Table 5.2 represents the resulted evaluation matrix. the only score for validation is optimal, because the domain knowledge is presence as a part of the domain expertise. However, the scores for time-consuming and quality are sub-optimal, because the ontology struc-

| | Protégé | SWOOP |
|---|---|---|
| Time-Consuming | low | low |
| Quality | high | medium |
| Validation | low | low |
| Post-Processing | high | high |

Table 5.1: Evaluation matrix for the composition tools and presuming ontology expertise

ture knowledge is not present and, therefore, a domain expert is unable to construct the ontology based on the interest of the domain. We believe that the presence of the ontology knowledge will increase the time-consuming and quality score and, therefore, decrease the intensive post-processing procedure.

| | Protégé | SWOOP |
|---|---|---|
| Time-Consuming | high | high |
| Quality | low | low |
| Validation | high | high |
| Post-Processing | high | high |

Table 5.2: Evaluation matrix for the composition tools and presuming domain expertise

## 5.2 Transformation Tool

As presented in section 4.5, the transformation tool Figure 5.4 shows an example of such transformation rules to generate the OWL 2 components which are relevant for constructing the semantic description of the domain. In the following two sub-sections, the evaluation result of the domain expert and ontology engineer is discussed.

### Ontology Expertise

Table 5.3 represents the resulted evaluation matrix. both scores for time-consuming and quality are acceptable, however, the score for validation is sub-optimal. The reason behind those results lies in the fact that the ontology engineer is not able to interpret the meaning of the data and its schema in case no enough description is available. Therefore, an intensive post-processing procedure is required to adjust the ontology and create axioms that satisfy the domain. We believe that the presence of the domain knowledge will increase the validation criterion score and, therefore, decrease the post-processing score.

### Domain Expertise

Table 5.4 represents the resulted evaluation matrix. the only score for validation is optimal, however, the scores for both time-consuming and quality is sub-optimal. The reason behind those results lies in the fact that the presence of the domain knowledge can interpret

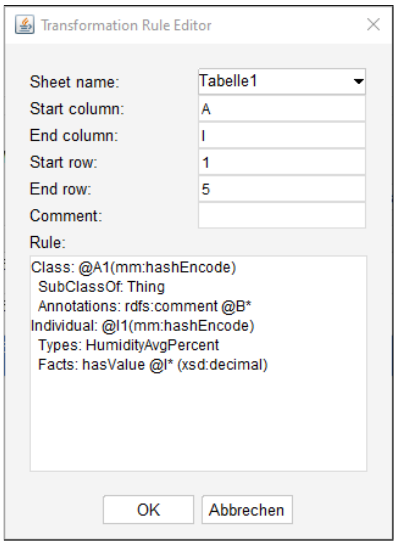


Figure 5.4: An example of transformation rules in Protégé

| | Protégé | Tarql |
|---|---|---|
| Time-Consuming | medium | high |
| Quality | medium | medium |
| Validation | low | low |
| Post-Processing | high | high |

Table 5.3: Evaluation matrix for the transformation tools and presuming ontology expertise

the intended meaning of the data. We believe that the presence of the ontology knowledge will increase the time-consuming and quality score and, therefore, decrease the intensive post-processing procedure.

| | Protégé | Tarql |
|---|---|---|
| Time-Consuming | high | high |
| Quality | low | low |
| Validation | high | high |
| Post-Processing | high | high |

Table 5.4: Evaluation matrix for the transformation tools and presuming domain expertise

## 5.3 Matching Tool

The matching tool uses the constructed domain ontology by the composition tool 5.1 and the constructed ontology from the data by the transformation tool 5.2 to find correspondences between their elements. Figures 5.5 and 5.6 shows a significant increase of the

matching result after adding additional annotations and axioms to its elements. In the following two sub-sections, the evaluation result of the domain expert and ontology engineer is discussed.

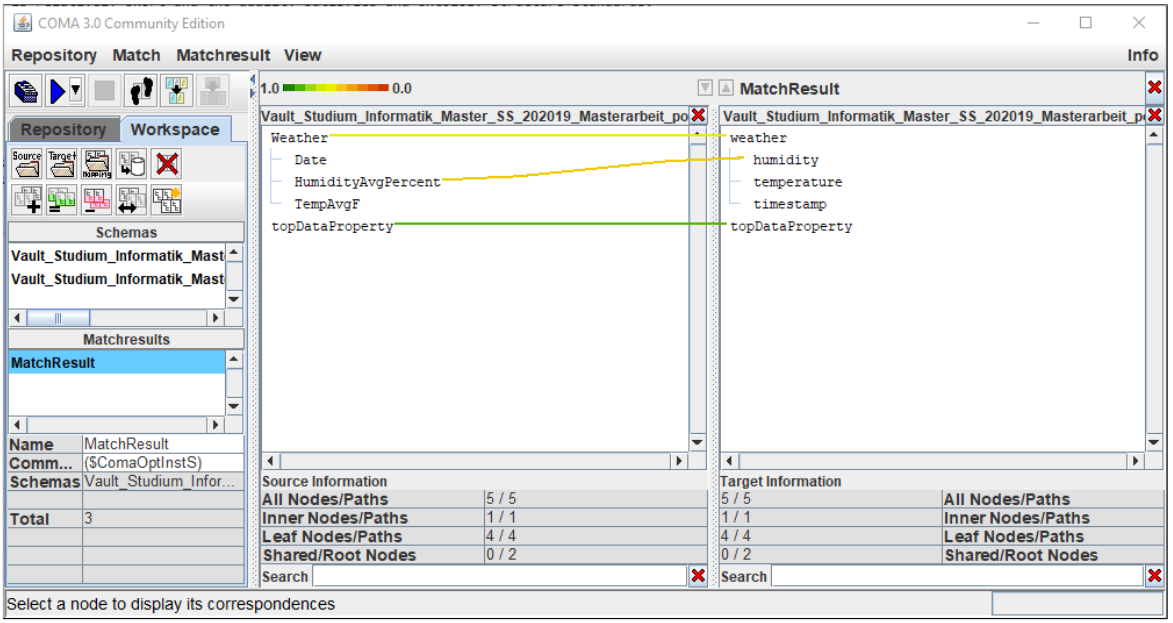


Figure 5.5: COMA matching result for not informative ontologies

### Ontology Expertise

Table 5.5 represents the resulted evaluation matrix. both scores for time-consuming and quality are acceptable, however, the score for validation is sub-optimal. The reason behind those results lies in the fact that the ontologies are well-defined due to the presence of the ontology knowledge; however, the validation of the matching result is not available without the presence of the domain knowledge. Therefore, an intensive post-processing procedure is required to adjust the matching result and confirm, decline, or add matching association between the ontologies' elements. We believe that the presence of the domain knowledge will increase the validation criterion score and, therefore, decrease the post-processing score.

| | COMA | AML |
|---|---|---|
| Time-Consuming | low | low |
| Quality | high | high |
| Validation | low | low |
| Post-Processing | medium | medium |

Table 5.5: Evaluation matrix for the matching tools and presuming ontology expertise

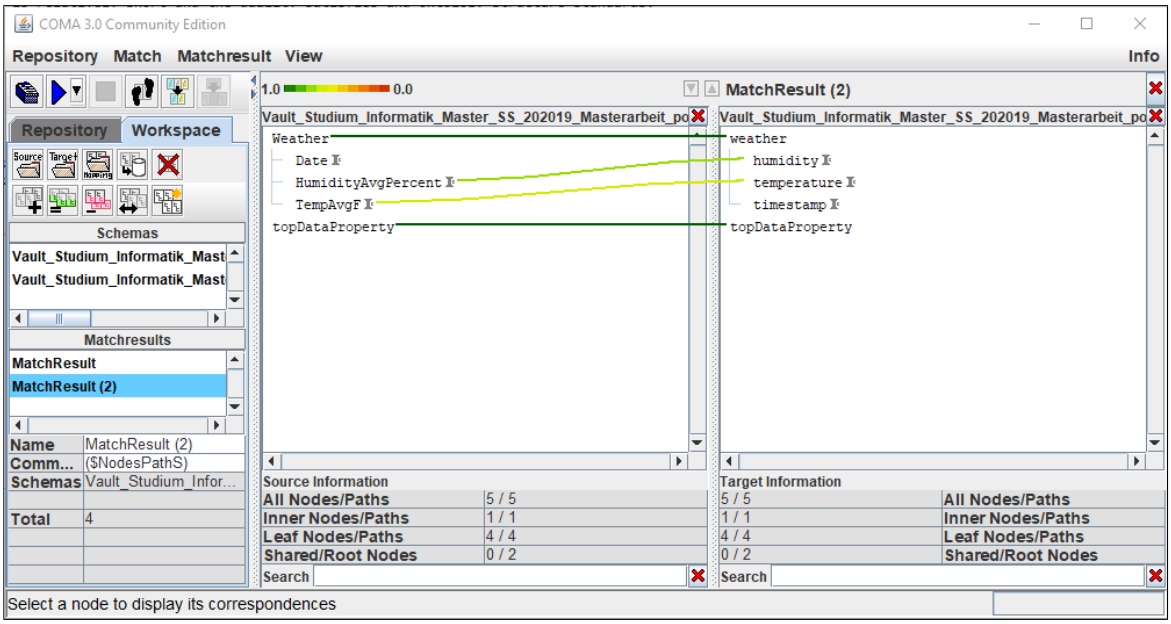


Figure 5.6: COMA matching result for informative ontologies

### Domain Expertise

Table 5.6 represents the resulted evaluation matrix. both scores for time-consuming and validation are optimal, however, the scores for both quality is sub-optimal. The reason behind those results lies in the insufficient ontology knowledge to construct informative ontologies and obtain high accuracy matching results. We believe that the presence of the ontology knowledge will increase the quality score by finding a meaningful matching association between the ontologies' elements and, therefore, decrease the intensive post-processing procedure.

| | COMA | AML |
|---|---|---|
| Time-Consuming | low | low |
| Quality | medium | medium |
| Validation | high | high |
| Post-Processing | medium | medium |

Table 5.6: Evaluation matrix for the matching tools and presuming domain expertise

## 5.4 Merging Tool

The merging tool uses the alignment results produced by the matching tool 5.3 and construct a new ontology that contains all elements from both ontologies and the correspondences between their elements.

### Ontology Expertise

Table 5.7 represents the resulted evaluation matrix. both scores for time-consuming and quality are acceptable, however, the score for validation is sub-optimal. The reason behind those results lies in the fact that an ontology engineer is unable to validate the merged ontology against the domain. Therefore, an intensive post-processing procedure is required to adjust the merged ontology to fit the domain specification. We believe that the presence of the domain knowledge will increase the validation criterion score and, therefore, decrease the post-processing score.

| | Protégé | OntoMerge |
|---|---|---|
| Time-Consuming | low | medium |
| Quality | high | high |
| Validation | low | low |
| Post-Processing | medium | medium |

Table 5.7: Evaluation matrix for the merging tools and presuming ontology expertise

### Domain Expertise

Table 5.8 represents the resulted evaluation matrix. both scores for time-consuming and validation are optimal, however, the scores for both quality is sub-optimal. The reason behind those results lies in the fact that a domain expert is unable to construct informative ontologies which affect the matching results and, therefore, the merged ontology. We believe that the presence of the ontology knowledge will increase the quality score and produce an informative merged ontology and, therefore, decrease the intensive post-processing procedure.

| | Protégé | OntoMerge |
|---|---|---|
| Time-Consuming | low | medium |
| Quality | medium | medium |
| Validation | high | high |
| Post-Processing | medium | medium |

Table 5.8: Evaluation matrix for the merging tools and presuming domain expertise

# 6 Conclusion

The purpose of this research was to investigate challenges and limitations in the domain description and ontology development. The main objective is to approach the domain engineering as a solution for bridging the gap between the domain experts and ontology engineers. The study proposed a domain knowledge development life cycle toolchain for the proposed domain engineering approach. This chapter summarizes the results achieved by this work and answers the research questions opened in section 1.3, besides presenting suggestions for future research works.

## 6.1 Achieved Goals

This study approached the domain engineering as a concept for bridging the gap between domain experts and ontology engineer. It also conducted extensive literature and related work review on two well-known fields of study: the syntactic- and semantic-based development approaches. The conducted research highlighted common strategies and algorithmic approaches to deliver the best results. In order to represent a domain of interest on semantic bases, this work introduced a toolchain with the following functionalities: ontology composition, data transfer, ontology matching, and merging.

A criteria-based decision matrix evaluation was conducted in chapter 5 to evaluate the proposed toolchain. The tools were tested based on the following criteria: time-consuming, quality, validation, and post-processing. The reason behind choosing those criteria is to indicate whether a domain expert or ontology engineer can achieve the best result without being experienced in the other field. For every functionality presented in the proposed toolchain, a couple of tools were tested against the criteria.

Based on the evaluation conducted on the proposed tool chain, the proposed research question in section 1.3 could be answered as follows.

1. The answer to the first research question, regarding the core functionality required for the development of domain knowledge on semantic bases, would be composition, transformation, matching, and merging.

2. The answer to the second research question, regarding the presence and collaboration of domain experts and ontology engineers in the domain knowledge development, would be yes. Although all tools in the suggested domain knowledge toolchain have a GUI, the fact that the inadequacies of domain knowledge for the ontology engineers, and the ontology knowledge for the domain experts, cause inaccurate results by the toolchain and requires an extensive post-processing procedure.

3. The answer to the third research question, regarding the existence of a tool which makes the domain expert or the ontology engineer role dispensable during the domain knowledge development, would be no. This answer can be implied from the one provided for the second research question.

## 6.2 Future Work

This research work is concluded by outlining a few future research suggestions. One of the proposed research suggestions is to integrate the different tools into one common interface. Protégé offers a plugin-based system to help the researcher and developer implement additional functionality. Another research proposal is to use the output resulted from the toolchain to integrate data from different sources but in the same domain. The toolchain can be further developed to generate a mediation schema to query the data sources and translate the results.

# Bibliography


[AB13] Sunitha Abburu and G. Suresh Babu. *Survey on Ontology Construction Tools.* International Journal of Scientific & Engineering Research, 2013.

[AH11] Dean Allemang and James A Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL.* Morgan Kaufmann/Elsevier, 2011.

[Bra13] Max Bramer. *Principles of Data Mining.* Springer, 2013.

[BRB11] Zohra Bellahsene, Erhard Rahm, and Angela Bonifati, editors. *Schema Matching and Mapping.* Springer, 2011.

[CDL+01] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. *Data Integration in Data Warehousing.* World Scientic Publishing Company, 2001.

[CFMV11] Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Gaia Varese. *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution.* Springer, 2011.

[Cho06] Chun Wei Choo. *The Knowing Organization: How Organizations Use Information to Construct Meaning, Create Knowledge, and Make Decisions.* Oxford University Press, 2006.

[Cim06] Philipp Cimiano. *Ontology Learning and Population From Text: Algorithms, Evaluation and Applications.* Springer, 2006.

[Dat10] Data Administration Management Association. *The DAMA Guide to the Data Management Body of Knowledge.* Technics Publications, LLC, 2010.

[DHI12] AnHai Doan, Alon Halevy, and Zachary G Ives. *Principles of Data Integration.* Morgan Kaufmann, 2012.

[DS15] Xin Luna Dong and Divesh Srivastava. *Big Data Integration.* Morgan & Claypool Publishers, 2015.

[Ehr07] Mark Ehrig. *Ontology Alignment - Bridging the Semantic Gap.* Springer, 2007.

[ES13] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching.* Springer, 2013.

[Eva14] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2014.

[GDD09] Dragan Gašević, Dragan Djurić, and Vladan Devedžić. *Model Driven Engineering and Ontology Development.* Springer, 2009.

[GLH15] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining.* Springer International Publishing, 2015.

[Gru93] Thomas R Gruber. *A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition.* Knowledge System Laboratory, Stanford University, CA, 1993.

[Hen01] James Hendler. *Agents and the Semantic Web.* IEEE Educational Activities Department, 2001.

[HJ12] Dawn E. Holmes and Lakhmi C. Jain. *Data mining: Foundations and Intelligent Paradigms: Volume 1: Clustering, Association and Classification.* Springer, 2012.

[HMS01] D J Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining.* MIT Press, 2001.

[HN04] Pavol Hell and Jaroslav Nešetřil. *Graphs and Homomorphisms.* Oxford University Press, 2004.

[Jab15] Fadi Jabbour. *Model Transformation for Domain Specific Architecture Languages in the Automotive Software Development.* 2015.

[KW09] Stanisław Kozielski and Robert Wrembel, editors. *New Trends in Data Warehousing and Data Analysis.* Springer, 2009.

[Ler16] Israël César Lerman. *Foundations and Methods in Combinatorial and Statistical Data Analysis and Clustering.* Springer Berlin Heidelberg, 2016.

[Maj14] Zoran Majkić. *Big Data Integration Theory: Theory and Methods of Database Mappings, Programming Languages, and Semantics.* Springer, 2014.

[MPB+12] Boris Motik, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Hasse, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. *OWL 2 Web Ontology Language Profiles Structural Specification and Functional-Style Syntax (Second Edition).* World Wide Web Consortium, 2012.

[MS02] Alexander Maedche and Steffen Staab. *Ontology Learning for the Semantic Web.* Springer US, 2002.

[Pau11] Heiko Paulheim. *Ontology-based Application Integration.* Springer, 2011.

[RA19] Meera Ramadas and Ajith Abraham. *Metaheuristics for Data Clustering and Image Segmentation.* Springer Berlin Heidelberg, 2019.

[RBSC+13] Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin. *Domain Engineering.* Springer Berlin Heidelberg, 2013.

[Ree13] April Reeve. *Managing Data in Motion: Data Integration, Best Practice Techniques and Technologies*. Elsevier, 2013.

[Row07] Jennifer Rowley. *The wisdom hierarchy: representations of the DIKW hierarchy*. Journal of Information Science, 2007.

[SCH+12] Michael Schneider, Jeremy Carroll, Ivan Herman, Peter F. Patel-Schneider, and Nuance Communications. *OWL 2 Web Ontology Language Profiles (Second Edition)*. World Wide Web Consortium, 2012.

[Vai14] Alejandro Vaisman. *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[WHSR16] Yong Wang, Dirk Herrling, Peter Stroganov, and Andreas Rausch. *Ontology-based Automatic Adaptation of Component Interfaces in Dynamic Adaptive Systems*. 2016.

[Wit14] Ian H. Witten. *Text Mining: From Ontology Learning to Automated Text Processing Applications*. Springer Berlin Heidelberg, 2014.

[ZC19] Lichun Zhang and Raymond L Chambers, editors. *Analysis of Integrated Data*. CRC Press, Taylor & Francis Group, 2019.